Wide analysis of QSlim Applied to Generated Convex Terrain

William Vickers-Hastings, Falmouth University, Games Academy

***Abstract*—QSlim has long been established as one of the most effective mesh simplification algorithms as it strikes a good balance between compute time and quality. There is however a lack of research on how it performs on convex terrain generation when compared to the terrain generation simplification (TGS), furthermore there is a lack of research in how QSlim scales with input mesh size and reduction rate. This paper addresses both these deficiencies and results show that in all practical scenarios QSlim does better job at optimising meshes on this type of terrain. It is not universally better, in extreme conditions QSlim breaks down and performs slightly to substantially worse than TGS. Extreme conditions are starting meshes with a very low poly count and/or a very high reduction rate (reduction of greater than 80% of original mesh size triangle count). With this terrain generation algorithm, such a low poly count start mesh yield very poor looking results even with no simplification applied, making QSlim's poor performance here irrelevant, it is still noteworthy. The larger the input mesh and the lower the geometry being taken away the better QSlim does.**

***Index Terms*—Quadric Mesh Simplification, Mesh Decimation, Retopology, Procedural Mesh, Terrain Generation, QSlim, Polygonal Mesh Simplification**

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 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 large grid-based terrain meshes is challenging as detail in the mesh is lost as polygon count is reduced.

An intuitive way to optimise height map based terrain meshes is to sample height map less frequently as shown in figure 1. This 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.

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. For processes like 3d sculpting, an extremely large mesh is often needed for artistic control [10], but such a high–resolution mesh is unnecessary for the final product, hence they are simplified once sculpting is completed [10].

Applying such an algorithm to a high sample-rate terrain mesh should produce a higher detail result with the same amount of geometry compared to running the terrain generation at low sample-rate with equivalent geometry size to the simplified mesh. - Per vertex and triangle, mesh simplification should produce a more accurate result.

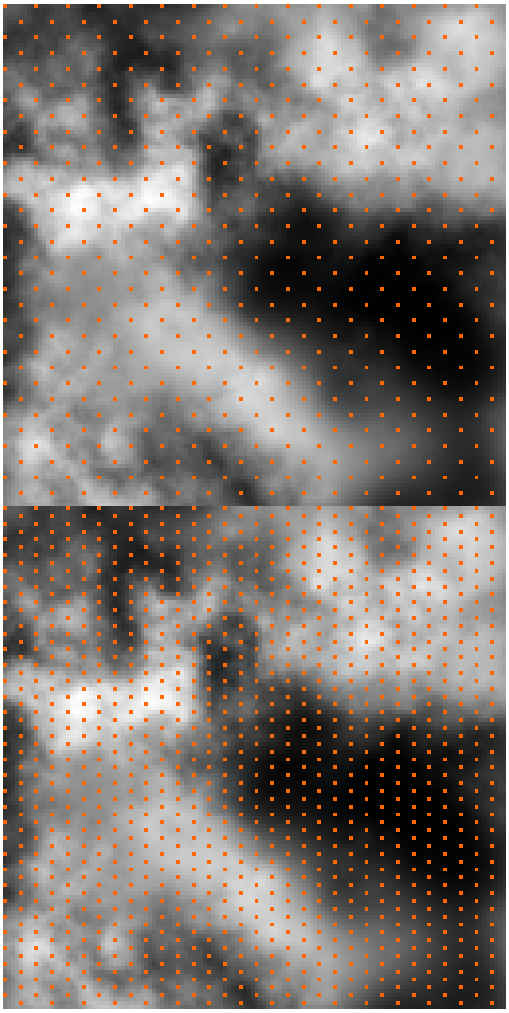

Fig. 1. This figure illustrates sampling the same height map at different resolutions. Each point becomes a vertex on a mesh with depth into/out of the page equal to the value sampled on the height map texture. The fewer samples the less detailed the final mesh will be but the less geometry also.

While mesh simplification is a well-established tool in 3D modelling [11]–[14], there is a lack of research on how well it performs as it's input parameters change. The two main input parameters for mesh simplification are the mesh to simplify and the amount to simplify it by.

This paper investigates how, Heckbert and Garland's tri-

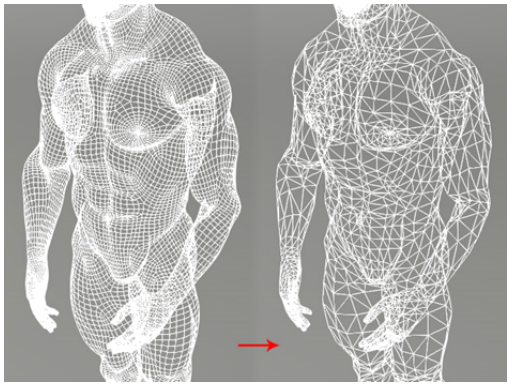

Fig. 2. Example mesh simplification of a character model from Character Creator 3 [11] Additional geometry in the mesh that provides minimal detail can be removed to reduce the number of vertices and triangles in the model.

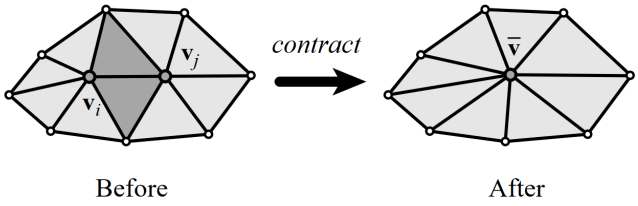

Fig. 3. Example of a single half edge contraction to simplify 10 triangles into 8, from figure 1 of [15]

angular mesh simplification algorithm, QSlim's [15] quality of result vary as increasing and decreasing the size of the input mesh and how a range of reduction rate effects the quality at each input mesh size. This paper also investigates how QSlim performs on simplex noise [16] based convex terrain generation meshes and compares it to the terrain generation algorithm configured to a comparable triangle count to evaluate if QSlim provides a better–quality result over low-resolution sampling of the height map.

The two simplification techniques being tested;

- Technique A, Terrain Generation Simplification (TGS). This technique uses the terrain generator to sample the same height map at a low resolution to produce a simplified mesh.
- Technique B, QSlim, proposed by Michael Garland and Paul S. Heckbert in *Surface Simplification Using Quadric Error Metrics* [15] which will simplify a high–fidelity mesh produced by the terrain generator.

II. Contextual Analysis

A. Mesh Simplification

To simplify a mesh is to maintain the overall shape and preserve as many details as possible of the mesh with less vertices and triangles (geometry or polygon count).

Two benefits of small meshes:

- **Data size**. Smaller meshes take up less space on the disk - meaning smaller game files and in they also take up less space in system memory and the 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 [15], [17]–[23], with one of the most cited and widely used algorithms known as QSlim [15] proposed in the 1997 because its strikes an excellent balance of quality and compute cost [24].

QSlim removes vertices from the mesh using a process known as half-edge collapse (see figure 3).

A half edge is a vector pointing towards another vertex - only edges (two 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 shape. [15] The algorithm contracts vertices with the lowest cost until it reaches an end condition, usually a target triangle count. [15]

Algorithms developed since [3], [19]–[21], [23], [25]–[28] nearly universally use the same theory of edge contraction and QEM as a base, both CPU and GPU algorithms.

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 [17], [29] additionally there is a lack of research on how an simplification algorithms perform with a large range of input mesh sizes and reduction rates. Many papers [3], [15], [19]–[21], [23], [25]–[28] only test and compare to each other using 3 or 4 levels of reduction rate and do not test different sizes of the same input mesh.

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. It's also possible algorithms like QSlim perform poorly on the type of mesh used for height-map based terrain generation. These meshes start as a large grid on a plane and unlike character meshes, they aren't enclosed, there is lose geometry along the edges of the grid. Why there may be a lack of research on this wide analysis of how simplification algorithms behave in different scenarios is possibly down to the time cost in running enough tests and creating the same model at different resolutions to test with Higher detail results should be achievable using simplification algorithms as they can keep more polygons in high detail areas of the mesh while removing them from flat areas (see figure 4), while in a uniform grid density found in simple height-maps (see figure 5) maintain a constant number of vertices per square unit of distance meaning waisted vertices exist in flat areas.

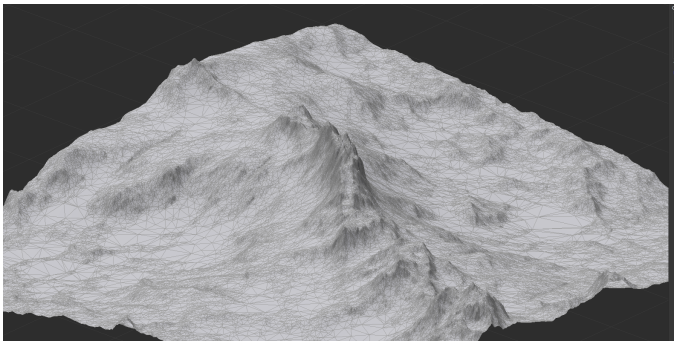

Fig. 4. Optimisation that reduces geometry in flat areas and puts the extra vertices into extreme elevation areas. Produced in World Machine [30]

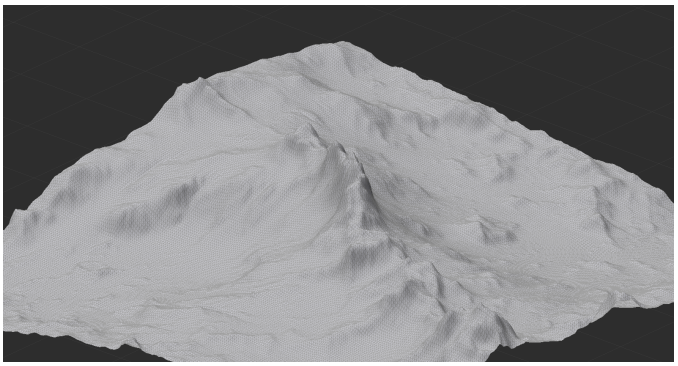

Fig. 5. Uniform grid terrain generation vertices are displaced only vertically more extreme height changes lead to larger triangles while flat areas have unnecessary geometry. Produced in World Machine [30]

B. Quantifying Results

To quantitatively measure the quality of a simplification algorithm, the geometric error of the simplified mesh compared to the original mesh is calculated using several methods, geometric deviation [31], [32], Hausdorff distance [33] and/or Root Mean Squared [15], [26].

Geometric Deviation is widely used [23], [25], [27], [28] and is the method chosen for this paper due to ease of implementation.

1) Geometric Deviation: 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. [15], [28], [31], [32] Averaged all these distances together produces the mean geometric deviation a metric that is used to quantify the difference between two meshes. [31]

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. [31] Figures 6 & 7 are the authors attempts to visualise this process. Because Geometric Deviation is calculated for each vertex in the larger mesh a heat map, like in figure 8, can be created to highlight areas of greater difference.

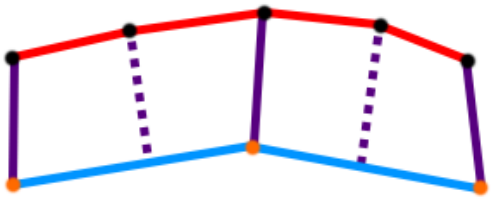

Fig. 6. 2D example of Geometric Deviation. The upper line is made of 5 vertices, the lower 3. The geometric deviation is found by finding the closest point on the lower line to a vertex on the upper line is. For the 2 vertices that exist in only the top line, the closest point is along the edges between vertices in the lower line, shown by dotted lines. The mean geometric deviation for these two lines is equal to the average euclidean length of the purple lines. For illustrative purposes the lines are separated, but when running the calculating they would be on top of each other.

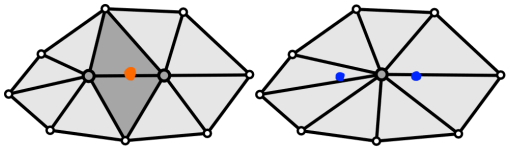

Fig. 7. This alternate view shows the points on the planes of each shape that geometric deviation would measure to. Assuming that the outside vertices perfectly match, the euclidean distance between the inside vertices and the coloured points is the geometric deviation for each vertex. Original Diagram based on figure 3

C. Quadric-Based Polygonal Surface Simplification (QSlim)

This algorithm simplifies a mesh by make a series of edge contractions. What edges contractions it makes is determined by an estimated quality impact the contraction will have on the mesh. This quality metric is called Quadric error (Q). Error Quadric are calculated as a matrix

1) Compute Quadric Error Matrices for all vertices
2) Select all valid edges
3) Compute optimal edge contraction for each valid edge. The optimal edge contraction has the lowest quadric error (Contraction cost), which is calculated by summing the error quadrics at V1 and V2.
4) Create a priority queue of contractions ordered by cost, sorted lowest to highest.
5) Dequeue the top contraction and contract that edge.
6) Update all quadric costs involving vertices in the contraction, including the vertex resulting from the contraction
7) repeat until an end condition is met, as outlined in section II-C1

1) Possible End Conditions:

- No more valid contractions
- Given face count is reached (where given face count $<$starting face count)
- Given vertex count is reached (where given vertex count $<$starting vertex count)
- Given number of edge contractions made (number of contractions >0)
- Given unit of time elapsed
- Cost of contraction exceeds given max (cost >0)

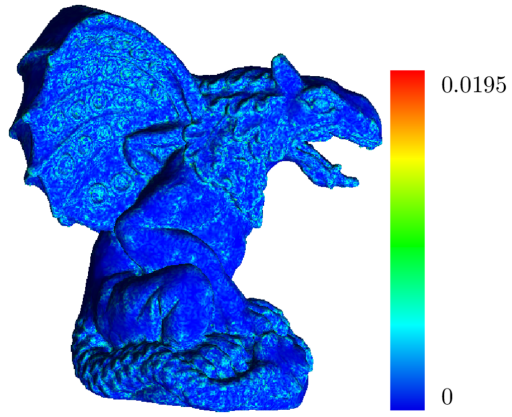

Fig. 8. Visualisation of mean geometric deviation of the gargoyle model used to test the GPU simplification algorithm implemented by Mousa and Hussein [28]

D. Convex Terrain Generation

The type of terrain generation being used to assess the algorithms is similar to height-map generation [34] but based on a subdivided Goldberg Polyhedron split into 162 tiles, 151 hexagons and 11 pentagons which are subdivided to a desired level of fidelity. Vertices are displaced in their local up direction, pointing away from the centre of the shape perpendicular to the tile they are in. The resultant mesh is convex.

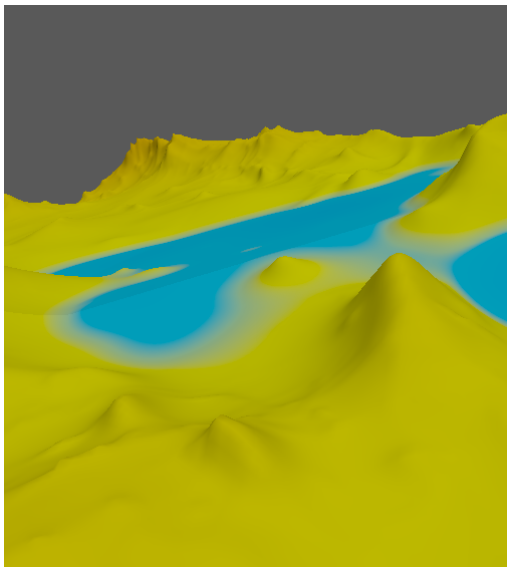

Fig. 9. Example terrain output from the computing artefact[1]

The terrain generator uses simplex-noise [16] in multiple layers with various settings based off work by Sebastian Lague [35], [36], combined with the gradient-trick for erosion proposed by inigo quilez [37]. This generator was created by the author in a custom Vulkan [38] framework in C# with SDL3 [39]

III. Scientific Questions & Hypotheses

Simplification algorithms are designed to work on arbitrary meshes pre-modelled in software like Blender [12], or Maya [13]. But little research exists on how these types of algorithms perform in a wide range of circumstances, how they behave as the input mesh fidelity changes and as the reduction rate changes.

As there is also a lack of research on simplification algorithms on procedural terrain geometry, this paper analyses QSlim's performances using simplex-based terrain generation algorithm which produces a convex terrain mesh.

This paper will compare the QSlim algorithm to low-fidelity terrain geometry generated by the terrain generation algorithm. The algorithms are implemented in the C# Vulkan environment.

This then all leads to the research question:

From a high fidelity generated convex terrain mesh, does the mesh simplification algorithms QSlim produce a low mean geometric deviation than a low-sample rate terrain generation pass in a wide range of input mesh fidelities and reduction rate scenarios.

Hypotheses:

1) QSlim always has a lower geometric deviation than a low fidelity generated mesh of comparable size regardless of input mesh fidelity or the requested reduction rate.
2) The Requested reduction rate will have the biggest impact on geometric deviation across both mesh simplification techniques.

IV. Methodology

A. Initial Design

1) Generate the same terrain at multiple fidelity levels
2) Run QSlim on these terrains with a wide range of simplification levels
3) Measure how long QSlim runs for
4) Use the Terrain generator to create terrains at similar simplification levels to that created by QSlim.
5) Measure how long the Terrain Generator runs for
6) Use Geometric Deviation to measure the difference between QSlim and the original mesh
7) Use Geometric Deviation to measure the difference between the Terrain Generator technique and the original mesh
8) Record all the measured data to a CSV file for analysis later.

To deliver this artefact, agile [40] development practices were used to rapidly iterate on the application. Tasks were kept track of using GitHub's task-board [41]. As it was iterated, the artefact was tested using unit tests, Integration testing and acceptance testing. It was refactored based on static code analysis [42] and functional decomposition. General software design came from the author's intuition and prior experience from literature like Clean Code [43].

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. This research does not breach any elements of any section in the BCS code of conduct [44]

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) Data Management: Data was recorded to six dictionaries internally to the computing artefact;

- Per Tiles Terrain Generator (Full data)
- Per Tiles Quadric Simplification (Full data)
- Per Planets Terrain Generator (Summary)
- Per Planets Quadric Simplification (Summary)
- Execution Time Terrain Generator (Summary)
- Execution Time Quadric Simplification (Summary)

For an example of what the first four dictionaries record. The first four dictionary are output in the format show in appendix A, figure 25, while the last two use the format in figure 26. Data was stored in the git repo[3].

The artefact exports these dictionaries to multiple CSV files after all tests have completed. These files are split into 3 groups. Execution time per planet, per planet data (Summary data) and per tile data (full data set). Two variants of each file are created, one is ordered by subdivisions then by reduction rate, the other ordered by reduction rate then by subdivisions. The summary files are much smaller and easier to work with for preliminary analysis. The full data set files were used to conduct data analysis presented in this paper.

2) Data Analysis: To test hypothesis 1, a T test per experiment scenario was run. An aggregate T test was run over the whole data set. The aggregate T test was run first to refute the null hypothesis before additional tests were performed. Cohen's D was also calculated for each T test. To test hypothesis 2 a linear regression test was performed over the whole data set to find the largest effect on quality. See Appendix F for the specific R code.

1) To refute the null hypotheses of hypotheses 1, some difference between the two simplification techniques must be demonstrated in all scenarios. To support the alternate hypotheses QSlim must have a low mean geometric deviation in each scenario. This was assessed with a T test for each scenario.
2) To support the alternate hypotheses of hypotheses 2, a linear regression model was run to find the most influential independent variable. The null Hypotheses will be refuted if any of the independent variables have an influence on mean geometric deviation.

D. Computing Artefact

To test the hypotheses a computing artefact that implements and can measure the performance of the two simplification techniques, QSlim and Terrain Generator Simplification (TGS), was created. As well as this it had to also generate a convex hull terrain mesh at multiple resolutions and with repeatable but randomizable results.

1) Terrain Generator: An existing terrain generating artefact was used, created by the author, data recording and QSlim implementation were added into framework[1].

2) (Sub) Divide and Conquer: To generate input meshes of varying size to perform experiments on, a way of subdividing a mesh with a high degree of control is required. The computing artefact already utilised a method to do this using a triangular subdivision algorithm. The existing algorithm did not meet the required level of granularity however as it scaled triangle count quadratically, so a new algorithm was developed.

This algorithm works by added new vertices along the edge of each existing triangle in a mesh, then interpolating and triangulating new geometry as shown in figure 10. This

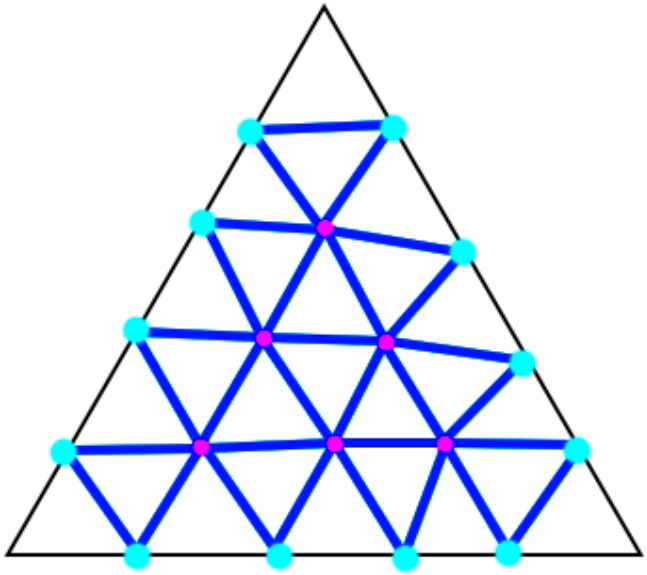

Fig. 10. Subdivision of a single triangle occurs in 3 main steps. This diagram shows 4 Subdivision steps. First (turquoise) new vertices are linearly interpolated along each edge of the triangle, equal to the number of subdivisions being made. Second (purple) additional vertices are linearly interpolated inside the triangle so everything can be triangulated. Finally (blue) new edges are added to triangulate the result, finishing the subdivision

algorithm offers a high degree of control over the number of triangles that can be added to a mesh, the equation that defines this is in figure 11. Ten subdivision levels and how many triangles this works out to can be seen in figure 12.

3) Geometric Deviation: The computing artefact needed a way to compute geometric deviation. This was achieved by translating the geometric deviation calculator written by

[1]Repository: https://github.falmouth.ac.uk/GA–Undergrad–Student–Work–24–25/Comp–302–2202796.git

$$TrianglesPerFace = x^2 + 2x + 1$$

Fig. 11. The equation that describes the total number triangles per face that will result for any subdivision level, where x is the number of subdivisions.

Michaël Roy [31], [45] from C++ into the C# Framework. This algorithm was modified to utilise multi–threading by the author. This modification was tested and no difference in results were observed, but a decent speed up was. The experiment utilised the multi–threading variant to reduce experiment duration.

4) QSlim: For QSlim the artefact uses a modified implementation developed by David Kuo, UnityMeshDecimation [46] which is a variant of the implementation of QSlim used in MeshLab [14] translated to C# for use with the Unity Game Engine. The author ported this package to run inside of the custom Vulkan C# environment, which involved writing a mesh conversion utility to convert meshes used by computing artefact to the internal mesh format used inside of UnityMeshDecimation as the included utility to convert unity meshes to this format does not work in the artefact as its not Unity. This implementation was selected as it was easy to setup (actually worked without major alteration) and based on a well–tested implementation of QSlim.

5) Execution Time: To measure execution time of each technique, the C# System.Diagnostics.Stopwatch [47] was used. To measure the Terrain Generator technique, the time to generate the low fidelity mesh was measured and recorded. To measure QSlim's execution time, the simplification of all meshes in the planet was measured and then added to the time it took to generate the high-fidelity mesh QSlim simplified. Theoretically, QSlim's execution time will always be longer than the terrain gen technique as a high-fidelity mesh must be generated which takes longer, then QSlim needs to run which is an additional step that TGS doesn't require.

E. Development Stages

1) Iteration 0: Terrain Generator: The first iteration of the software artefact was the terrain generator that lies at the heart of everything being tested. This was created as part of another module and is clearly labelled in the repo.

2) Iteration 1: Prototype: The artefact needed a way to measure the difference between any two meshes (Geometric Deviation) to analyse how similar in shape they are. For this prototype, the author translated Michaël Roy's MeshDev [31], [45] from C++ to C# and integrated it into the software Framework. Some simple acceptance testing was performed by generating two planets from the same seed at different fidelity levels then using the translated MeshDev implementation to calculate geometric deviation between them and visualise the output. An issue became apparent that the Subdivision algorithm used in the original terrain generator did not have enough granularity in mesh size control.

3) Iteration 2: Subdivision Upgrade: This iteration focused on upgrading the subdivision algorithm used by the terrain generator to gain more granular control over the mesh size as outlined in section IV-D2, this is crucial for the experiment. Additionally, the MeshDev implementation was improved slightly by adding multi-threading to it. This sped up its calculation time considerably and did not affect results.

4) Iteration 3: QSlim: QSlim was then implemented and tested using the Stanford Bunny and ran as expected. This took a whole iteration to complete as finding a version of QSlim that worked in the software framework took some time as did integration.

5) Iteration 4: Experiment Setup: With all parts of needed to run the experiment present and tested, the experiment itself was then set up as outlined in IV-G. This involved creating ways to store data created by the MeshDev implementation and measuring the execution time of QSlim and the terrain generator. Several sets of data gathering occurred at this point as a form of acceptance testing, bugs where discovered, fixed and then the application was run again.

6) Iteration 5: Polish: The final artefact polish involved adding a CLI interface to allow a user to change the experiment configuration (Data output director, number of planets, subdivision levels etc), view the result for a specific planet, and to verify the QSlim and MeshDev were working as expected. The CLI was tested with Unit tests.

F. Testing

Evidence of testing can be found in Appendix C

1) Geometric Deviation: To ensure the geometric deviation implementation is being accurately calculated it was compared to its source paper and source code [31], [45] and tested using a sphere and a cube.

2) QSlim: To ensure QSlim was providing outputs in line with its author's expectations, it was tested using the Standford Bunny [48] which QSlim is demonstrated with [15]. The implementation of QSlim used in this research differs slightly from Garland's original algorithm, it is able to take into account additional vertex attributes like vertex normals or UVs when computing quadric errors as such the performance will not be exact to the original paper but should be enhanced. This idea was proposed by Garland in the original paper [15], [49].

3) Additional Notes: General Artefact testing. The CLI interface for the artefact was tested using Unit tests. Application stability is important for this experiment as the algorithms involved take a long time to run. For tracked time the artefact ran for a total of 10.1hrs just generating and simplifying meshes over two test runs (190 tests x 2). Geometric Deviation calculation adds additional overhead of about 6x. In total

the application ran for roughly 60hrs across two sessions during which no crashes or errors occurred. Thought develop the artefact has been refactored, splitting its user interface and data collection/export modules into separate classes from the experiment management file. Additionally, techniques like method extraction, decomposition, and abstraction utilising features inside of Visual Studio 2022 [50] like static code analysis to guide improvement.

G. Final Experiment

Level	Subdivisions	Total (162)	Per Hex (151)	Per Pent (11)
Base shape		960	6	5
1	5	34 560	216	180
2	10	116 160	726	605
3	15	245 760	1 536	1 280
4	20	423 360	2 646	2 205
5	25	648 960	4 056	3 380
6	30	922 560	5 766	4 805
7	35	1 244 160	7 776	6 480
8	40	1 613 760	10 086	8 405
9	45	2 031 360	12 696	10 580
10	50	2 496 960	15 606	13 005

Fig. 12. This table shows the number of triangles at each subdivision level tested and the size of the base shape. A planet is made of 162 tiles, 11 Pentagons and 151 Hexagons. Each tile is a separate mesh. These values where calculated using the formula in figure 11

Level	Target % original geometry to keep
1	95%
2	90%
3	85%
4	80%
5	75%
6	70%
7	65%
8	60%
9	55%
10	50%
11	45%
12	40%
13	35%
14	30%
15	25%
16	20%
17	15%
18	10%
19	5%

Fig. 13. This table shows the selected reduction levels each simplification technique was tested at, expressed as a percentage of original triangles remaining.

The variables being measured about each technique is primarily mean geometric deviation, the difference in shape compared to a high–fidelity mesh and secondarily the execution time of the simplification technique. Figure 14 provides a high–level guide for experiment operation, for detail breakdown see Appendix D.

To ensure confidence in results, 10 different seeds were used to in testing. The framework supports this because uses a seed value which terrain generation is based off, the seed used to generate each planet was recorded so results can be visualised and if required repeated for validation. The seed will produce the same results across multiple environments, provided all terrain generation settings remain constant, which was true during the experiment

To simplify the meshes using the QSlim, it needs an additional input, how much to simplify the high-fidelity mesh by. This is given as a % of the original Triangle count. Each algorithm was tested using a set of reduction levels shown in figure 13.

To compare the QSlim to low fidelity terrain, each seed had a low fidelity terrain mesh generated which has a roughly equal triangle count to the reduction levels above. This can't always be exact due to limitations with the subdivision algorithm. QSlim's target reduction level is set equal to the actual reduction rate the terrain generator was able to achieve to ensure comparability of results.

V. Results & Findings

A. Results

After the computing artefact was implemented, data was gathered and analysed using R, Appendix F, the results of which can be found in the repo[2]. To view the raw data sets, see the repo[3]. Graphs were plotted using Seaborn [51] and Pandas [52] in Python 3.9 [53], see Appendix B for heatmaps and additional plots.

1) Summary Data: In Figure 15 the summary test showed that across all experiments on average, group 1 (QSlim) outperformed group 0 (Terrain Generator) in providing the lowest geometric mean deviation in simplified geometry. Further statistical significance was reached as p <2.2e-16 owing to the amount of data gathered across all the experiments (1620 observations per technique per experiment). At the overall measured effect size of 0.178, 1161 samples where needed to be statistically significant.

Figure 16 showed the pratt analysis of the linear model run on the whole data set to rank the independent variables by importance (effect on Mean_Dev). The technique used has the lowest effect of the three, with Input_Reduction having the largest effect.

2) Detailed Data: Figure 19 & 20 Show Geometric Mean Deviation measured at in each experiment. From these graphs it is clear that QSlim is not always the best technique for minimising Geometric Deviation with this terrain generator. It always performs worse than the terrain generator at Input Reduction = 5% when the Subdivision count reaches 15 and above. Below 15 subdivisions, it frequently swaps places with the terrain generator. Many of these results contain

[2]Data analysis can be found in the repo at Data/Test–13/Data_Analysis.txt. https://github.falmouth.ac.uk/GA–Undergrad–Student–Work–24–25/Comp–302–2202796/ blob/main/Data/Test–13/Data_Analysis.txt

[3]Dataset csvs can be found in the repo at Data/Test-13. https //github.falmouth.ac.uk/GA–Undergrad–Student–Work–24–25/Comp–302–2202796/tree/main/Data/Test–13

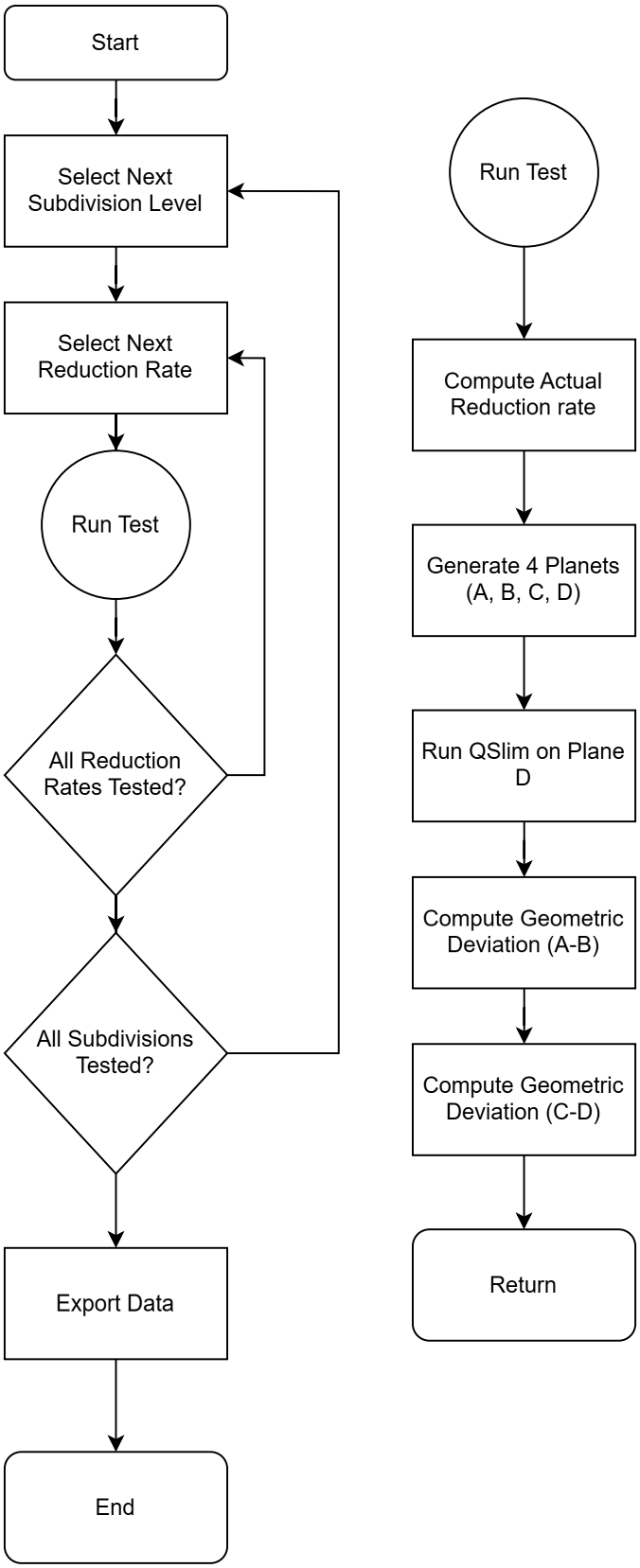

Fig. 14. A test is run at each reduction rate for each subdivisions level until all have been tesed. The left column shows how the main loop works, while the right breaks down a single test. Note in the detailed list of Appendix D step 3, planet B is generated at a different subdivision level as this is the TGS result.

Welch Two Sample t-test		
t = 44.686	df = 606423	p-value <2.2e-16
alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0		
5 percent confidence interval: 7.051545e-05 7.698498e-05		
mean in group 0	mean in group 1	
0.0004043668	0.0003306166	
Overall Cohen's D		
variable	Mean_Dev	
group 1	Terrain Generator	
group 2	QSlim	
effect size	0.178	
n1	307800	
n2	307800	
magnitude	negligible	

Fig. 15. Overall T test and Cohen's D for the whole data set of 615600 observations. This test shows that the results have statistical significance, and that group 1 (QSlim) has a lower Mean_Dev than Group 0 (terrain generator). The effect size over the whole data set is negligible at 0.178. The minimum effect size for an individual experiment was 0.234, which requires 673 samples per technique for statistical significance, 1620 samples per technique per experiment where collected[2]. At the overall effect size (0.178), 1161 samples for each technique were needed to reach statistical significance, 307800 where collected in total for each technique.

Pratt analysis of Linear Model product of the standardized coefficient and the correlation	
Response variable: Mean_Dev	
Total response variance: 4.205635e-07	
Analysis based on 615600 observations	
3 Regressors:	
Algorithm Src_SubDiv Input_Reduction	
Proportion of variance explained by model: 38.78%	
Metrics are normalized to sum to 100% (rela=TRUE).	
Relative importance metrics:	
	pratt
Algorithm	0.008337474
Src_SubDiv	0.472128750
Input_Reduction	0.519533776

Fig. 16. Pratt analysis of linear regression model[2]. Analysis of the 3 independent variables showed that Input_Reduction has the highest effect on Mean_Dev, followed by Src_SubDiv then the algorithm used. The P value of the linear regression test is <2.2e-16, meaning that although it has a small effect, Algorithm does have an effect.

duplicates. At 50% input reduction rate and above, (Figure 20) QSlim outperforms the terrain generator in every case with 5 exceptions where the experiment was unable to produce a valid mesh from the terrain generator at subdivisions = 5 for Input Reduction rates of 85-95%, and for subdivisions = 10 & 15 for Input Reduction = 95. In these cases, the experiment failed to run.

Figure 18 showed the T test result for all scenarios negative values in this matrix indicate the terrain generator scored a lower mean geometric deviation than QSlim. The magnitude of the T value showed size of the lead each technique held over the other. The NaN results are anomalous, these occurred

when the mean deviation for both techniques was equal to 0. The trend for this table showed that as Input Reduction rises, QSlim performs better than the terrain generator as shown by the T values turning positive and raising in magnitude. This all maps to figure 21 which categories mean deviation results differently, grouping by input reduction rate and plotting mean dev by subdivisions. The general trend showed that as subdivisions increase the lower mean deviation can be achieved at each input reduction step. There are some anomalies in the data, specifically in the plots of Input reduction = 5, 10 and 15 then in plots Input reduction = 85, 90 and 95. These are two different anomalies, the first showed mean dev getting worse, then improving with subdivision count, for QSlim and for the terrain generator, particularly in plot Input reduction = 5. The second anomaly surfaces when the terrain generator and QSlim do not simply the mesh at all, leading a mean geometric deviation of 0, as the input and output meshes are the same.

Figure 22 showed that in several cases at low input fidelity, the target reduction rate was not met by QSlim or by the terrain generator. In cases where both techniques failed to hit the target this is because a single vertex could not be divided further. In Figured 23 this behaviour is no longer present as there are enough vertices that each reduction rate can be met. Note in figure 22 & 23 that consistently throughout all results the input reduction rate of 5% was never achieved by QSlim, even in the high–fidelity mesh, Src_SubDiv = 50.
Figure 24 presents a closer look at all subdivision steps between Input_Reduction = 5–25 highlighting the cases where the Terrain generator and QSlim did not match, indicating QSlim failed to reach the target reduction rate. In Src_SubDiv = 5 & 10, in no scenario did QSlim reach the target. In Src_SubDiv = 15–30 QSlim is able to match the terrain generator in more scenarios each time Src_SubDiv increases. When Src_SubDiv >= 30, QSlim always matches the terrain generator except from at 5% Input_Reduction and as already shown in figure 22 & 23, QSlim was never able to hit the 5% target.

3) Execution Time: Figure 17 shows the additional time QSlim needed over the terrain generation on average in each scenario. As the number of subdivisions increase, the time taken increases. The less original geometry QSlim is allowed to keep also increases execution time.

	Average time increase over terrain generation in Seconds									
	Src_SubDiv									
Input_Reduction	5	10	15	20	25	30	35	40	45	50
5	0.27	1.10	2.60	4.89	7.96	11.99	16.96	23.20	30.46	39.14
10	0.27	1.10	2.61	4.81	7.88	11.71	16.74	22.74	29.78	38.35
15	0.27	1.10	2.58	4.75	7.69	11.37	16.07	22.02	28.99	37.23
20	0.27	1.09	2.53	4.66	7.38	11.00	15.75	21.37	28.05	36.23
25	0.27	1.09	2.43	4.55	7.20	10.81	15.09	20.88	27.27	35.28
30	0.27	1.05	2.34	4.42	7.02	10.34	14.47	20.10	26.39	33.59
35	0.26	1.05	2.33	4.34	6.80	10.11	14.19	19.38	25.44	32.44
40	0.26	0.98	2.23	4.10	6.57	9.49	13.46	18.40	24.31	31.18
45	0.26	0.99	2.11	3.94	6.33	9.14	13.02	18.15	22.99	29.77
50	0.26	0.90	2.11	3.72	6.08	8.77	12.57	16.75	21.70	28.20
55	0.26	0.90	1.96	3.50	5.80	8.45	11.69	16.20	21.00	26.62
60	0.23	0.82	1.97	3.52	5.50	8.04	11.15	14.89	19.47	24.77
65	0.23	0.82	1.81	3.27	5.17	7.63	10.68	14.23	18.57	23.81
70	0.23	0.82	1.81	3.00	4.83	7.20	10.02	13.57	17.85	25.49
75	0.23	0.71	1.62	3.00	4.47	6.73	9.52	12.18	16.18	20.83
80	0.23	0.71	1.63	2.73	4.47	6.26	8.93	11.42	15.30	18.84
85	0.17	0.71	1.44	2.73	4.13	5.80	8.35	10.77	14.44	17.95
90	0.17	0.71	1.44	2.46	8.47	5.82	7.77	10.11	12.81	16.90
95	0.21	0.58	1.23	2.46	3.77	5.35	7.22	9.43	11.95	14.88

Fig. 17. This matrix shows the average time in seconds QSlim took to compute it's result over the terrain generator for each scenario. The hardware used for his was an AMD Ryzen 7 7840HS (32GB DDR5-5600) and an RX 7700S (8GB GDDR6-18Gbps), the experiment was conducted on Windows 10 22H2, disconnected from the internet

T–Test Matrix	Src_SubDiv									
Input_Reduction	5	10	15	20	25	30	35	40	45	50
5	18.772	17.65	-24.519	-36.878	-43.947	-48.09	-44.428	-37.814	-28.753	-24.425
10	18.772	-5.7936	-33.622	-20.222	-16.603	2.696	4.7922	11.94	16.307	16.116
15	18.772	-22.139	-14.392	3.1506	12.536	16.475	19.431	20.945	21.977	22.345
20	-9.1065	-8.5639	9.2946	14.046	21.582	22.397	23.347	23.79	24.764	24.422
25	-9.1065	-8.5639	10.65	20.017	16.76	24.46	19.314	25.806	19.785	25.685
30	-9.1065	15.468	25.805	24.144	26.924	28.304	28.884	27.89	28.049	27.874
35	18.307	15.468	25.805	27.789	29.691	30.296	30.527	30.444	30.185	29.84
40	18.307	27.661	31.64	32.048	32.518	34.981	34.285	33.526	32.863	32.204
45	18.307	27.661	38.838	34.74	35.899	37.673	34.908	35.346	36.041	33.562
50	18.307	39.85	38.838	41.627	39.464	40.714	38.836	39.402	39.744	38.158
55	18.307	39.85	45.796	47.267	43.352	43.898	43.555	41.604	41.724	41.723
60	52.123	55.405	45.796	47.267	47.46	47.298	46.84	46.478	45.948	45.536
65	52.123	55.405	54.686	53.111	51.897	50.79	49.84	48.987	48.202	47.512
70	52.123	55.405	54.686	58.999	56.304	54.261	52.618	51.532	50.448	51.556
75	52.123	69.4	62.901	58.999	60.573	57.7	55.658	56.391	54.752	53.461
80	52.123	69.4	62.901	64.333	60.573	60.926	58.362	58.62	56.836	57.125
85	NaN	69.4	69.974	64.333	64.264	63.732	60.946	60.8	58.726	58.768
90	NaN	69.4	69.974	68.494	67.023	63.732	63.117	62.582	62.019	60.3
95	NaN	NaN	NaN	68.494	67.023	65.648	64.548	63.675	62.916	62.414

Fig. 18. T–Test Matrix showing the T result for each individual experiment. Tests which scored NaN showed no difference in mean, indicating the null hypothesis is true. Negative T values show the terrain generator outperformed QSlim, while positive values indicate QSlim outperformed the terrain generator, with the larger the value in either direction indicating a lower mean deviation. Out of 190 tests shown here 182 had p–values <2.2e–16, five failed due to Mean_Dev being 0, of the remaining 3, the highest p–value was 0.007 meaning other than in tests that outright failed, statistical significance was reached. Out of 190 T–tests, QSlim performed better than the terrain generator in 166, and performed worse in 19 and in 5 saw no difference. See the repo [2] for the full breakdown of the t—tests.

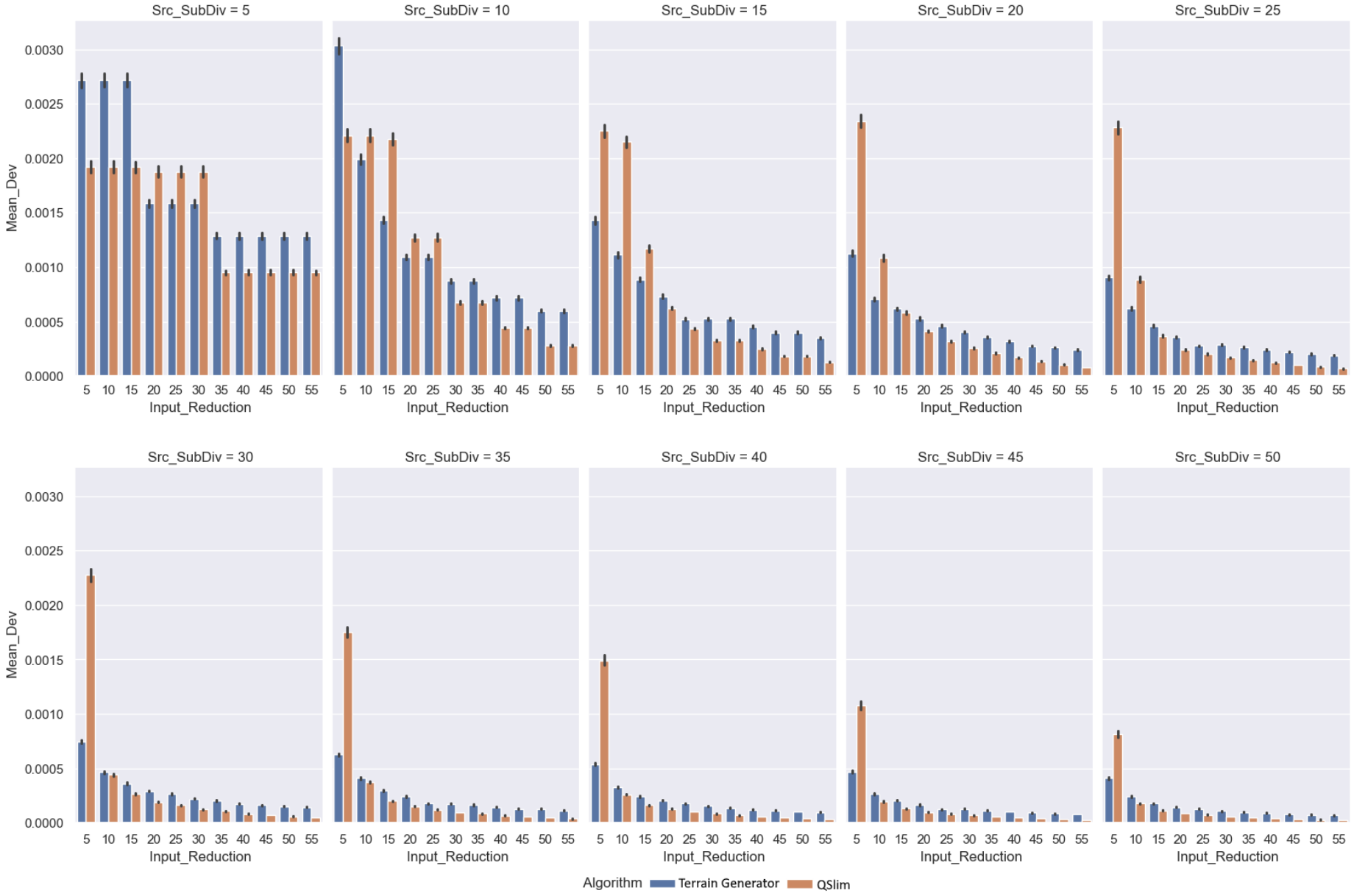

Fig. 19. Plots of all Subdivision levels, Input Reduction 5%–55%. Each graph showed how Input_Reduction affected Mean_Dev at a given subdivision level, Src_SubDiv. This graph showed that as the input mesh size increased, Mean Geometric Deviation fell for QSlim and the Terrain Generator. As Input Reduction nears 0% both techniques perform worse. At 5%, QSlim fails to to deliver a better result than the Terrain generator regardless of input mesh size. At 35% QSlim out performs the Terrain generator at every input mesh size tested.

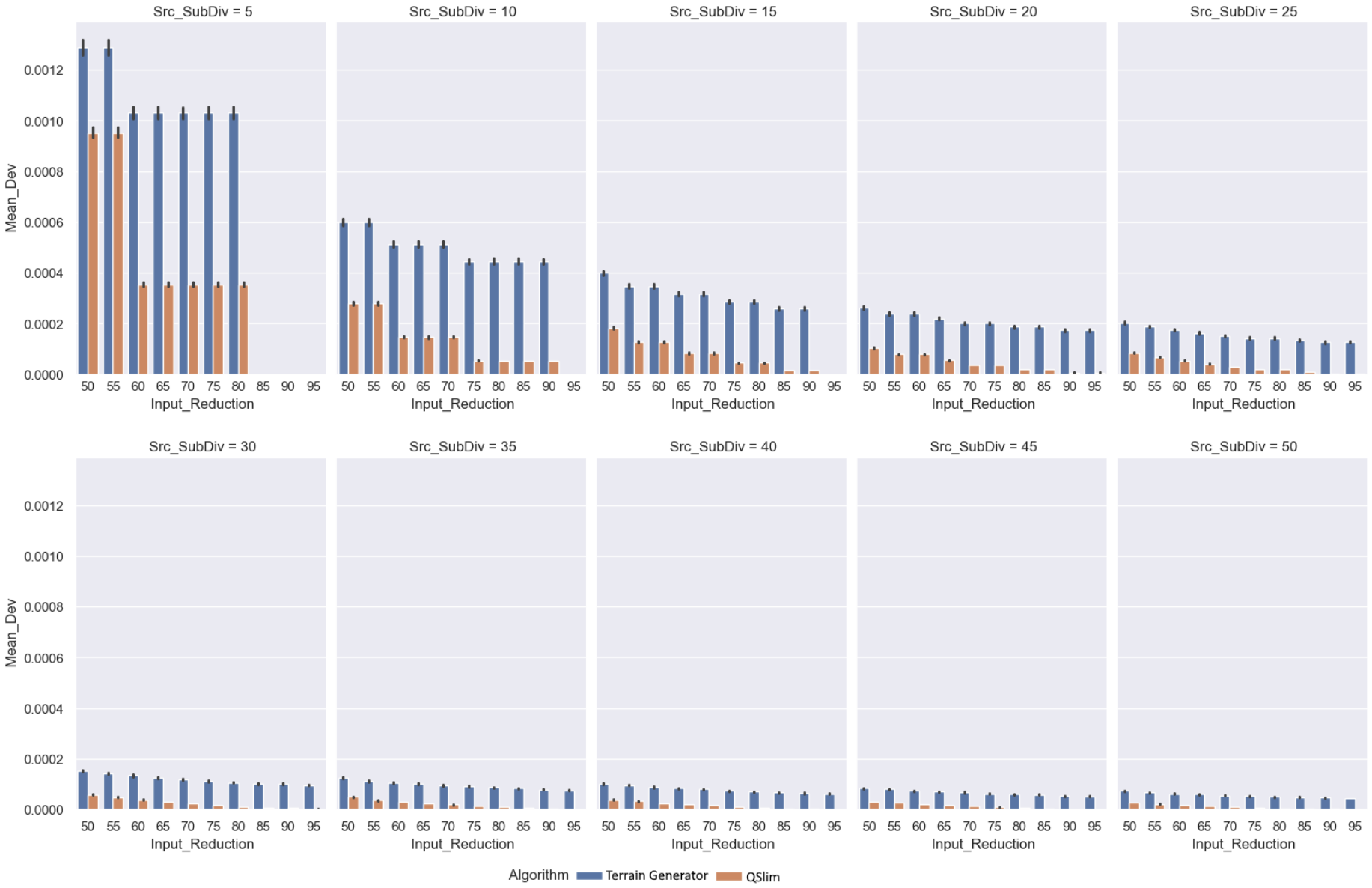

Fig. 20. Plots of all Subdivision levels, Input Reduction 50%–95%. Each graph showed how Input_Reduction affected Mean_Dev at a given subdivision level, Src_SubDiv. This graph showed that as the input mesh size increased, Mean Geometric Deviation fell for QSlim and the Terrain Generator. At 50%–95 Input Reduction and above QSlim outperforms the Terrain generator in every case. Note in cases where the blue **and** orange bars aren't visible the test failed. This occurred 5 times, Src_SubDiv = 5, Input_Reduction = 85%–95% (3), Src_SubDiv = 10 & Src_SubDiv = 15, Input_Reduction = 95% (1 + 1). In no other cases did tests fail in this figure or in figure 19

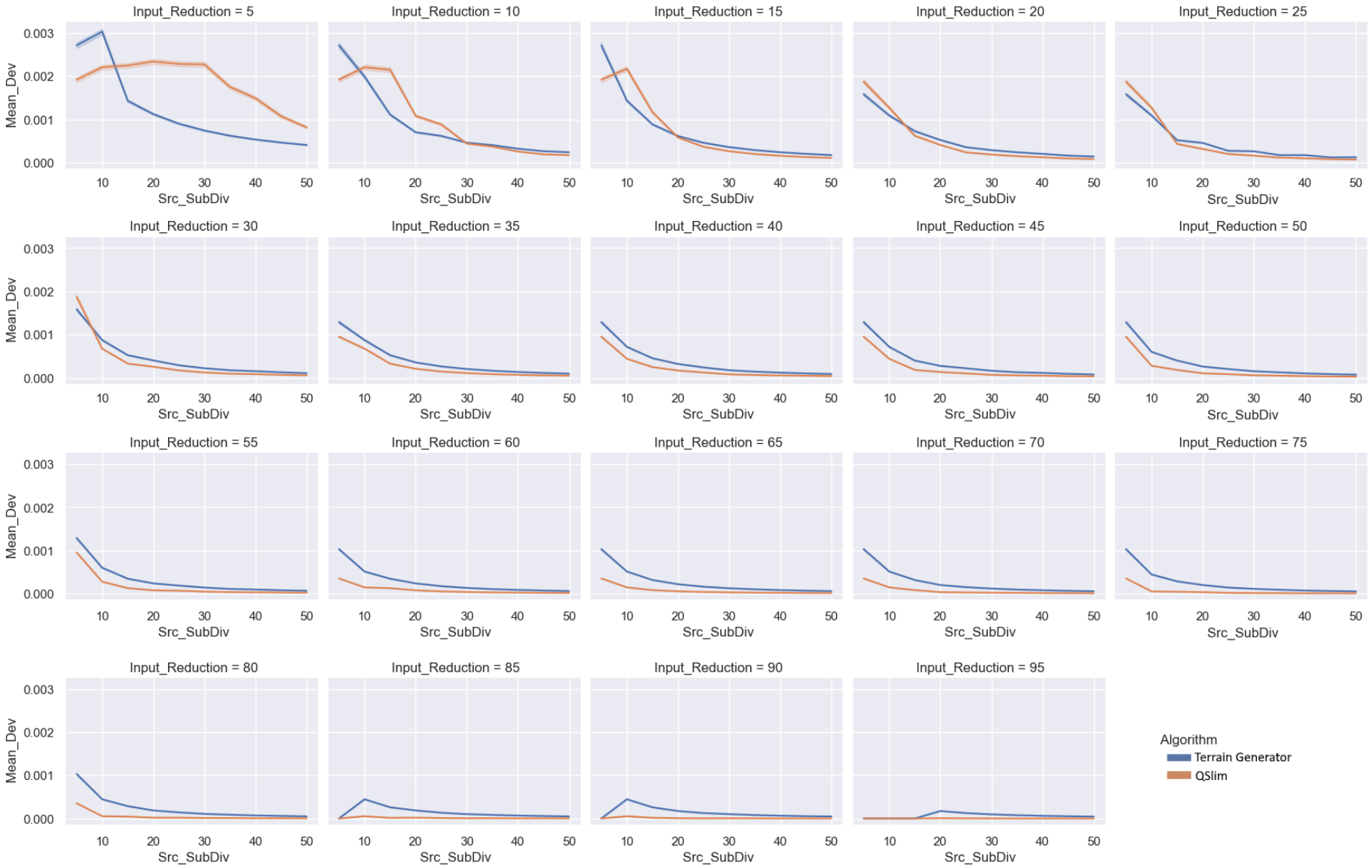

Fig. 21. Broadly maps with 18 showing the relationship between mean deviation and subdivisions categorised by input reduction %. Interesting behaviour can be seen in Input reduction = 5–15 where QSlim starts off better than the terrain generator then gets worse, before getting better again in Input Reduction = 10 & 15, while in 5 in never gets better but seems to trend in that direction if more subdivisions where tested. The T test NAN results can be mapped to mean dev = 0 when input reduction = 85–95 when subdivisions equal 5, 10 or 15.

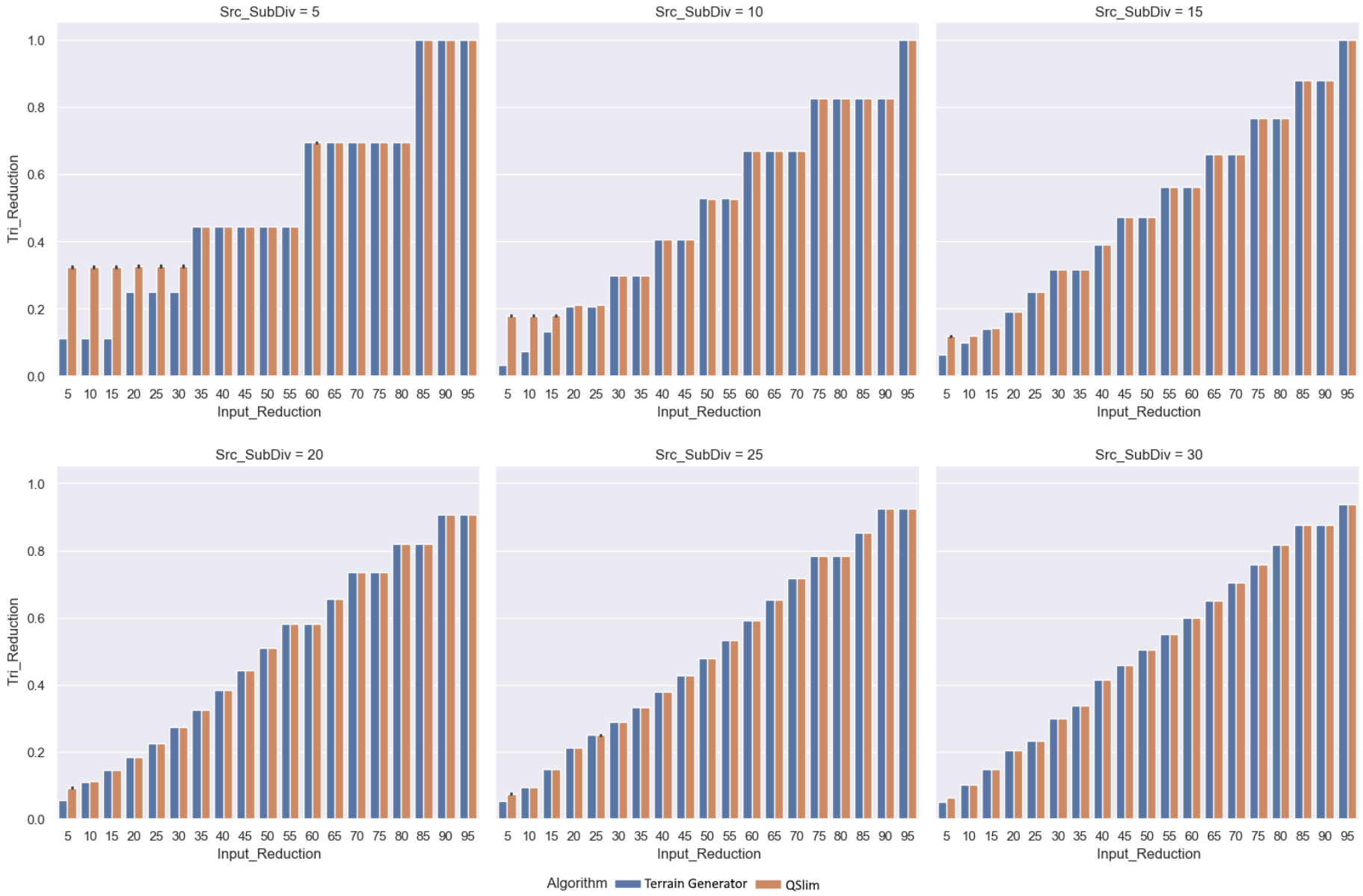

Fig. 22. Plots Input_Reduction against Tri_Reduction between 5% and 95% requested reduction and Src_SubDiv 5–30. This graph showed how successful each technique was at reaching the requested reduction rate. Until Src_SubDiv hits 20, the Requested reduction rate of 95% was never reached in all cases, a rate of 100% was used in these scenarios, indicating no difference from the high–fidelity mesh. All plots here have at least 1 duplicate reduction rate plotted. This showed situation where the application was unable to reduce the mesh to the target triangle count until a threshold reduction percentage was reached, creating the stair stepping effects seen clearly in Src_SubDiv 5–15, which slowly went away in 20–30 and as shown in figure 23 stopped all together.

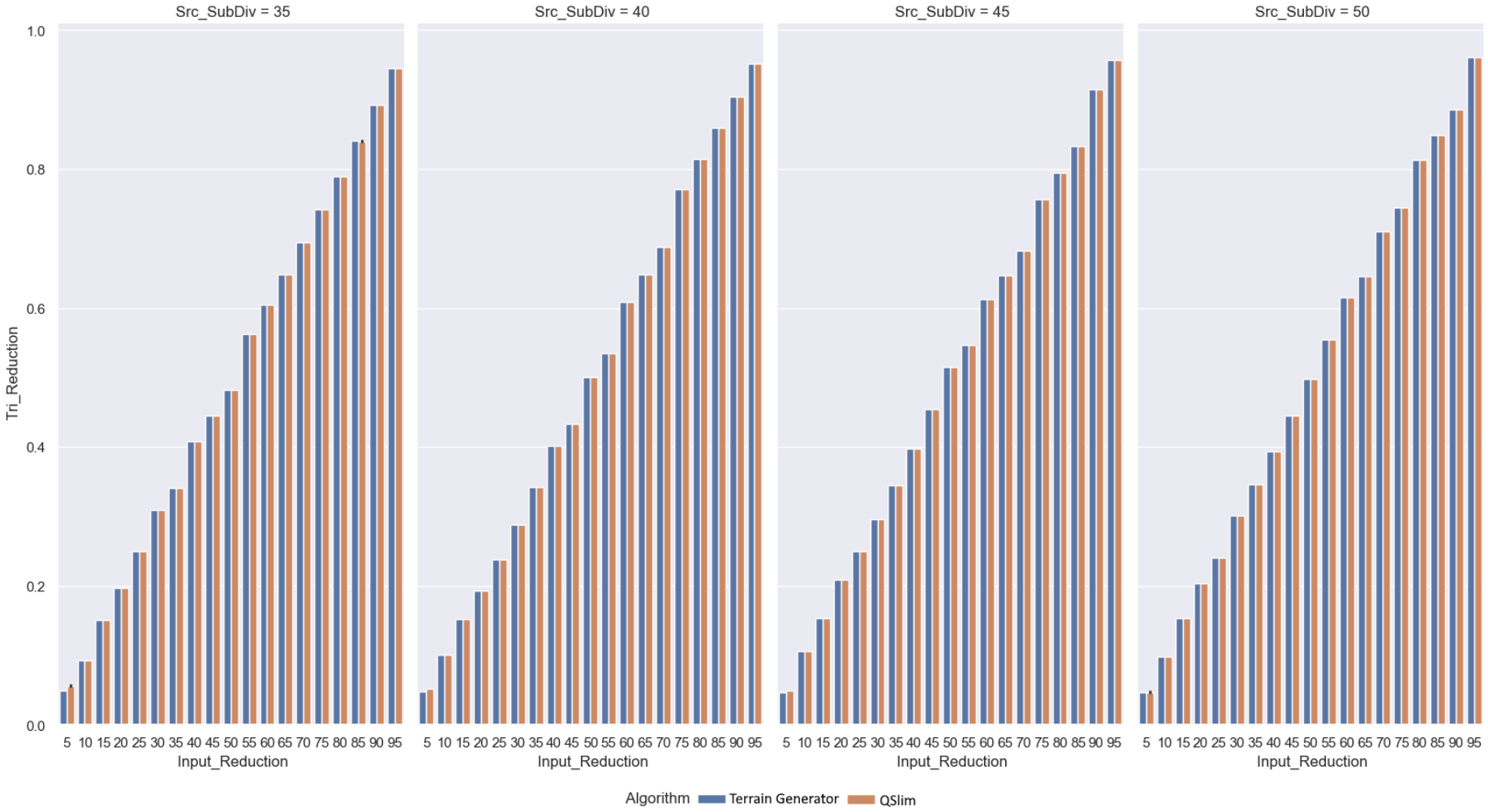

Fig. 23. Plots Input_Reduction against Tri_Reduction between 5% and 95% requested reduction and Src_SubDiv 35–50. This graph showed how successful each technique was at reaching the requested reduction rate. Unlike in figured 22, the requested reduction rate was reached by at least one simplification technique in all scenarios plotted here.

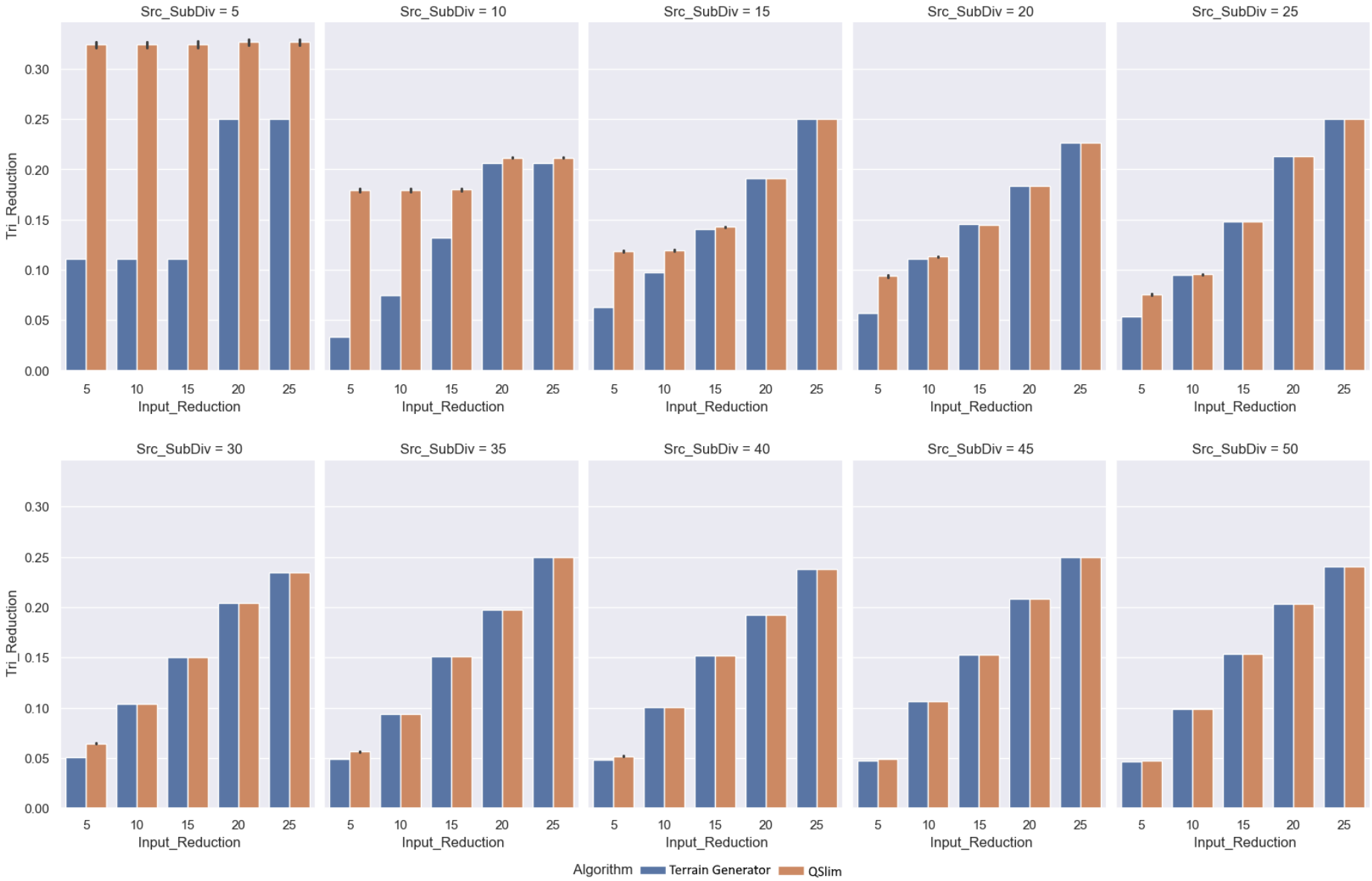

Fig. 24. Close look at Input_Reduction against Tri_Reduction between 5% and 25% requested reduction. This graph showed how successful each technique was at reaching the requested reduction rate. If QSlim is working correctly it should match the terrain generator exactly. All the way through Subdivisions = 5 to 50, requested 5% reduction rate, QSlim never matches the terrain generator and is always above it, even if by a tiny margin.

B. Findings

1) Hypotheses 1: Results of statistical T tests refute the null hypotheses for hypotheses 1. Figure 15 showed a statistically significant difference between QSlim and the Terrain Generator ($p<2.2e-6$). Hypotheses 1 is refuted by figure 18 as not every T test showed that QSlim is better than the terrain generator, what it does show is in 19 of 190 tests, QSlim is worse, 5 tests failed so in 166 scenarios, QSlim provides a lower geometric mean deviation. The lowest significance for the 190 individual t tests, excluding failed scenarios, was $p = .007$[2]. In all scenarios which passed (185/190) statistical significance was reached.

2) Hypotheses 2: Results of the linear regression test refuse the null hypotheses for hypotheses 2. Figure 16 show that all three independent variables had influence on mean geometric deviation. The statistical significance for this test was $p <2.2e-6$[2], meaning this test was statistically significant. The regression test refutes hypotheses 2, Src_SubDiv did not have the largest impact on mean geometric deviation, Input_Reduction did (51.95%). Src_SubDiv was able to account for 47.21% of the difference in mean geometric deviation, a significant portion still, but not as high as Input_Reduction. The technique used had a minor but measurable influence on mean geometric deviation of 0.83%. This relationship can be seen in figures 19 & 20 The subdivision count influences on mean geometric deviation, at very low values quite a large effect, but Input reduction is able to extract larger differences. There is a large difference between the technique used as shown by the plots, but this difference remains relatively constant throughout the data set, and QSlim has a negative impact on mean deviation when the input reduction rate is 5 or the subdivisions are low.

VI. Discussion

QSlim is used as a tool to simplify meshes in 3d modelling software, through simplification, quality is always lost and QSlim and its derivatives do their best to minimise it. With generated terrain meshes like those used in this paper, it is simple to create a simplified mesh by running the generator a lower fidelity. This paper shows although that may be simple to do, it can leave lead to a large loss in quality when compared to an algorithm like QSlim, in a large majority of scenarios using a wide range of input mesh sizes and target reduction rates, QSlim outperformed the terrain generator.

A. Major Findings

QSlim has not be had such a wide analysis of how it performs with such a wide array of input mesh sizes and reduction rates. The results show that Both have a large impact on simplified mesh quality (Mean Geometric Deviation), with the reduction rate having the largest impact (see figure 16) this is arguably in line with expectations, the quality of result is directly tied to the reduction rate as quality is always lost when geometry is removed, QSlim is an algorithm to minimise that quality loss, whether the quality loss is noticeable is another matter not answered by this paper, but nevertheless its desirable the minimise it. On the type of terrain generation used in this paper, QSlim is able to out significantly cut down the mean deviation compared to the terrain generation technique for simplifying the mesh.

1) QSlim's Weak areas: QSlim starts to breakdown as the input reduction rate really becomes extreme, 5% of original geometry is very extreme and, in all scenarios, QSlim was either unable to reach 5% of original geometry or was outperformed by the terrain generator. The opposite is also true however, the more of the original geometry QSlim is configured to keep the higher the quality is, QSlim is able to out perform the terrain generator in all reduction rates above 40%, and coupled with high input mesh sizes (25 Subdivisions+), QSlim is able to significantly out perform the terrain generator.
The results show QSlim does perform worse with low input mesh sizes. This also makes sense as with more input geometry there is more choice in the edges it can collapse and more data to better estimate the new positions of edges it collapses. This analysis showed it is possible to out perform terrain generator based simplification for terrain meshes using QSlim, it can be worth using it in such situations where high detail and reasonably low mesh sizes are needed. The drawback is performance. The Implementation for QSlim used in this paper could take over a minute to run on the larger mesh sizes and low reduction rates. This limits its application in real-time environments like video games. However, faster implementations exist, work has been done to port QSlim to run on the GPU and a number of papers have demonstrated significant performance improvements by doing so, but they are very complex and require use of graphics programming APIs like CUDA and OpenCL which may present a large barrier to entry. There are challenges in parallelising QSlim as separate threads could end up in race conditions if one thread collapses out a vertex another thread was using. These challenges have been addressed by a number of papers [3], [22], [24]–[28], [54], [55] which implement variants of QSlim for GPGPU compute and also some for CPUs.

2) Other types of terrain: Terrain meshes are not the only form of procedural geometry that QSlim could be applied to and future work should look to analyse its performance with them. Additionally, this type of terrain, which is simple convex height-map based terrain, is not the only type of terrain mesh, non-convex hull terrain generation, like that possible with Marching cubes [56] would be an interesting avenue to investigate QSlim's's performance as would different types of convex hulls where different kinds of terrain is produced. The Terrain used in this paper varies between flat planes representing seas and oceans, to low hills and sharp mountains as can be seen in appendix B. An analysis of each of these biome types may yield different results.

B. Results Outliers

1) Failed Tests and Duplicate Results: 5 tests failed to produce meshes different from the original high–fidelity mesh. the bottom 3 tests of Src_SubDiv = 5 and the last test in Src_SubDiv = 10 & 15, ranging from 85% to 95% of original geometry. The reason why these tests failed is down to a limitation with the terrain generator, which uses a subdivision algorithm to divide a shape up to varying levels of fidelity.

This algorithm works by adding vertices along the edges of each triangle within the base mesh. A subdivision level of 5 adds 5 new vertices along each edge in of a triangle in the mesh. Figure 11 describes the equation for that describes the curve for this x is the number of subdivisions. At low input resolutions (5–15), there is not enough granularity to subdivide a mesh that is equal to 95% of the original mesh, the closest the artefact was able to get to the requested percentages that failed to produce simplified meshes, was 100%. This is an integer rounding problem, it is not possible to subdivide the mesh to 4.5, it must be 4 or 5. This creates a stair stepping effect that can be seen in multiple places in the data as duplicate results and in figure 22. The closest the artefact was able to match the subdivided mesh to was sometimes equal to a step it had already made. When 35 subdivisions for the input mesh size was hit, enough granularity is gained to at least somewhat match the requested reduction rate, after this point each result is unique. This limitation of the terrain generator is not desirable for the experiment but is overall not very significant. In future work, generated meshes should have more granular geometry size control, especially at mesh small sizes.

2) QSlim Anomalies: QSlim showed several anomalies during testing. These anomalies were brought out by extreme conditions. Extreme conditions here being a combination of very low input mesh size (5–15 subdivisions) and very high reduction rates (5%–25%) In such conditions, QSlim performed terribly. In some of these scenarios, QSlim failed to hit the same reduction level as the terrain generator, the QSlim mesh always had more geometry. Additionally, when these cases arose, QSlim performed worse in geometric mean deviation than the terrain generator in all scenarios apart from 4. In every 5% scenario, across all subdivision steps, it failed to match the Terrain Generators reduction rate. The 4 outlines of outliners can be seen in figure 19, subdivisions 5 and 10, looking at these scenarios results in figure 24 QSlim held a significant geometry size advantage the terrain generator did not which explains why QSlim appeared to outperform the terrain generator. This can be explained by QSlim itself. QSlim only considers the edge pairs valid at initialisation time. It will never consider pairs it has created itself. This means at very low input mesh sizes, sometimes it cannot collapse enough edges to meet the target triangle count, before it runs out of valid pairs to collapse if the requested triangle count is extremely low, additionally as it has to collapse all the edges it can, edges that the algorithm has calculated will have very large impact on quality are collapsed when given more starting geometry the algorithm would never do. This is a noteworthy behaviour of the QSlim, however with this terrain generator configured to produce such small input meshes, results without any simplification, QSlim or otherwise are not of high visual quality anyway, fine details don't start appearing until the subdivision level hits 40–50, acceptable visual results start around 25 subdivisions.

3) Execution Time Anomalies: Two anomalies can be seen in the execution time matrix (Figure 17) that break the trend within their column. One at SubDiv = 50 Reduct = 70, the other at SubDiv = 25, Reduct = 90. These are almost certainly down to Windows doing operations in the background that effect hardware utilisation. They could be eliminated in the future by conducting additional test passes.

VII. Conclusion

The purpose of this paper was to analyse how QSlim performed on convex terrain geometry and to conduct a wide–ranging analysis of its performance as input mesh size and reduction rate varied as there is a lack of research on how QSlim behaves in all 3 of these areas. In a large majority of a wide range of input mesh size and reduction rate scenarios, QSlim produces a lower mean geometric deviation than terrain generation simplification (TGS). From this analysis it can be concluded that QSlim performs well on Convex Terrain, as its able to outperform the TGS technique in all reasonable scenarios in terms of mean Geometric Deviation. It has also been observed how QSlim performs with a wide range of input meshes sizes and reduction rates and can state that QSlim is able to do a better job with bigger input meshes and with lower reduction rates. QSlim is not better in *all* scenarios, but it is in the 166 out of 185 tests that passed. 5 tests that failed, due to limitations with the subdivider algorithm, not QSlim and as these tests where in very extreme unrealistic scenarios, it does not have a significant impact on the findings of this paper had they succeeded.

A. Recommendations

The CPU implementation of QSlim used in this paper, even on relatively powerful modern hardware at time of writing, is not fast enough to practically use in real-time situations such as in video games. It is recommended to implement a parallelised variant of QSlim either CPU or GPU especially would provide a significant performance improvement [**?**] which theoretically would be performant enough to use in games What can be concluded is that if there are the compute resources available to utilise QSlim, then a significant shape quality improvement will be observed for the same number of Polygons. QSlim should not be used in extreme scenarios. An extreme scenario is a very low input mesh size coupled with a very low or very high reduction rate. In the case of very low mesh sizes (Less than 1000 triangles), the value of QSlim adds in reducing geometry is either very minor or even has negative impact on quality. QSlim quickly breaks down at very low reduction rates. If there is a hardware scenario where 200 triangles is too many, QSlim is not the tool to solve the problem as the compute required to run QSlim would have an even larger impact on performance as presumably the hardware is very weak. the low value in extreme scenarios is true regardless of reduction rate, as the number of triangles saved by reducing a 1000 triangle mesh by 5% is 50, while at 1 million triangles its 50 000. Such a minor reduction with such a large mesh would provide game developers with a lot more headroom to add additional models or increase quality in other models than at 1000 tris, making its value low or negative with small models.

B. Future work

This work provided valuable insight in how QSlim behaves on convex terrain geometry, but it did not analyse more

complex types of terrain generation such as non-convex terrain generators like marching cubes. Seeing how voids within terrain are handled by QSlim would be a valuable piece of information as QSlim is able to handle non-convex 3d models with specific care put into this area [15].

This research represents a quantitative analysis of QSlim's ability to maintain shape accuracy as measured by geometric mean deviation, this does not mean qualitatively, the outputs appear pleasing or comparatively better than the TGS technique to a human and a wide ranging study to explore so should be undertaken. While this paper provided insight into a wide range of reduction rates and starting mesh sizes for terrain, it did not do so for traditional 3D models, validating QSlim behaves similarly in this area would be valuable but testing it would incur additional such challenges as getting origin models in a wide range of fidelities.

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] B. Simonds, *Blender Master Class: A Hands-On Guide to Modeling, Sculpting, Materials, and Rendering*. No Starch Press, Incorporated. [Online]. Available: http://ebookcentral.proquest.com/lib/falmouth-ebooks/detail.action?docID=1142865

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

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

[14] 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

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

[17] 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.

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

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

[20] 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

[21] 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

[22] 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

[23] 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

[24] Y. Song, R. Fellegara, F. Iuricich, and L. De Floriani, "Parallel topology-aware mesh simplification on terrain trees," vol. 10, no. 2, pp. 1–39. [Online]. Available: https://dl.acm.org/doi/10.1145/3652602

[25] 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

[26] 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

[27] 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

[28] 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

[29] 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

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

[31] 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

[32] 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

[33] 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

[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] Sebastian Lague. [unity] procedural planets (e07: ocean depth). [Online]. Available: https://www.youtube.com/watch?v=OULxvDLojic
[36] S. Lague. SebLague/procedural-planets. Original-date: 2018-08-16T15:20:06Z. [Online]. Available: https://github.com/SebLague/Procedural-Planets
[37] I. Quilez. Inigo quilez. [Online]. Available: https://iquilezles.org/articles/morenoise/
[38] Vulkan. Home | vulkan | cross platform 3d graphics. [Online]. Available: https://www.vulkan.org/
[39] SDL Community. SDL3/FrontPage. [Online]. Available: https://wiki.libsdl.org/SDL3/FrontPage
[40] 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/
[41] GitHub. About projects. [Online]. Available: https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects
[42] gewarren. Code analysis in .NET. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/fundamentals/code-analysis/overview
[43] Robert C. Martin, *Full details and actions for Clean code: a handbook of agile software craftsmanship*, 1st ed. Prentice Hall. [Online]. Available: https://www.vlebooks.com/Product/Index/865506?page=0&startBookmarkId=-1
[44] BCS Trustee Board, "Code of conduct for BCS members." [Online]. Available: https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf
[45] Michaël Roy. MeshDev - mesh comparison software. [Online]. Available: https://meshdev.sourceforge.net/
[46] D. Kuo. crescent3983/UnityMeshDecimation. Original-date: 2022-03-27T15:06:59Z. [Online]. Available: https://github.com/crescent3983/UnityMeshDecimation
[47] Microsoft. Stopwatch class (system.diagnostics). [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch?view=net-9.0
[48] Stanford University Computer Graphics Laboratory, "The stanford 3d scanning repository." [Online]. Available: https://graphics.stanford.edu/data/3Dscanrep/
[49] 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
[50] Microsoft. Visual studio 2022 IDE - AI for coding debugging and testing. [Online]. Available: https://visualstudio.microsoft.com/vs/
[51] M. Waskom, "seaborn: statistical data visualization," vol. 6, no. 60, p. 3021. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.03021
[52] Wes McKinney and Community. pandas - python data analysis library. [Online]. Available: https://pandas.pydata.org/about/
[53] Python Foundation. Python release python 3.9.0. [Online]. Available: https://www.python.org/downloads/release/python-390/
[54] Jie Tang, Gangshan Wu, Bo Xu, and Zhongliang Gong, "Fast mesh similarity measuring based on CUDA," in *2010 IEEE International Conference on Progress in Informatics and Computing*. IEEE, pp. 911–915. [Online]. Available: http://ieeexplore.ieee.org/document/5687883/
[55] S. M. Shontz and D. M. Nistor, "CPU-GPU algorithms for triangular surface mesh simplification," in *Proceedings of the 21st International Meshing Roundtable*, X. Jiao and J.-C. Weill, Eds. Springer, pp. 475–492.
[56] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," vol. 21, no. 4, pp. 163–169. [Online]. Available: https://dl.acm.org/doi/10.1145/37402.37422

ACKNOWLEDGMENTS

I would like to thank John Speakman for his help and support in the creation of this dissertations through suggestions regarding data visualisation.
I would also like to thank Dr. Michael Scott for his help with statistics and R/R Studio and for his dedication to this course ensuring its high standards and educational quality.
Finally, I'd like to thank Robert Lancaster for proof reading my terrible spelling.

APPENDIX A
COLLECTED DATA FORMAT

Seed	Tile_ID	Algorithm	Src_SubDiv	Input_Reduction	Vert_Count	Tri_Count	Vert_Reduction	Tri_Reduction	Min_Elev	Max_Elev	Mean_Elev	Min_Dev	Max_Dev	Mean_Dev
-238246973	0	0	5	95%	26880	103680	100%	100%	-0.07675676	0.14479381	0.012721919	0	0	0
1172812092	16	0	35	55%	2436	4374	57.75%	56.25%	-0.003561106	0.049596906	0.018747509	0	0.001941207	0.0001044

Fig. 25. Two rows picked from one of the tables of the full data set to illiterates the type of data collected for difference in shape.

Seed	Tile_ID	Algorithm	Src_SubDiv	Input_Reduction	Vert_Count	Tri_Count	Vert_Reduction	Tri_Reduction	ExeuctionTime
-238246973	0	0	5	95%	26880	103680	100%	100%	3.9642
1293553328	0	0	50	10%	146880	737280	11.1%	9.84%	5.2593

Fig. 26. Two rows picked from one of the tables of the execution time data set to illiterates the type of data collected to for execution time.

APPENDIX B
ADDITIONAL FIGURES

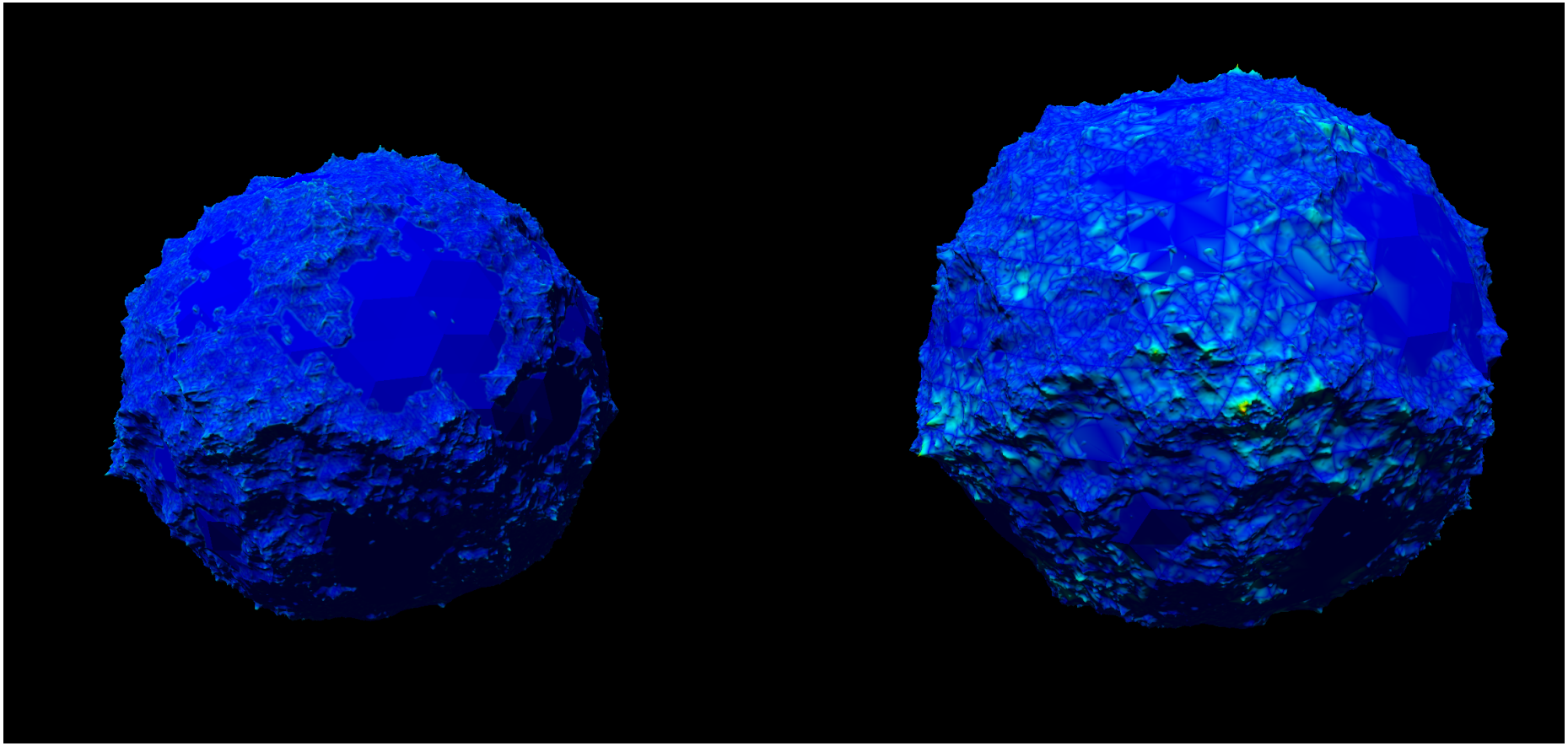

Fig. 27. TGS on the left, QSlim on the right. This shows the geometric deviation heat map for 50 subdivisions at 5% of original geometry. Here QSlim performs worse than TGS and that is reflected in the heat map with large areas of turquoise and some yellow and green.

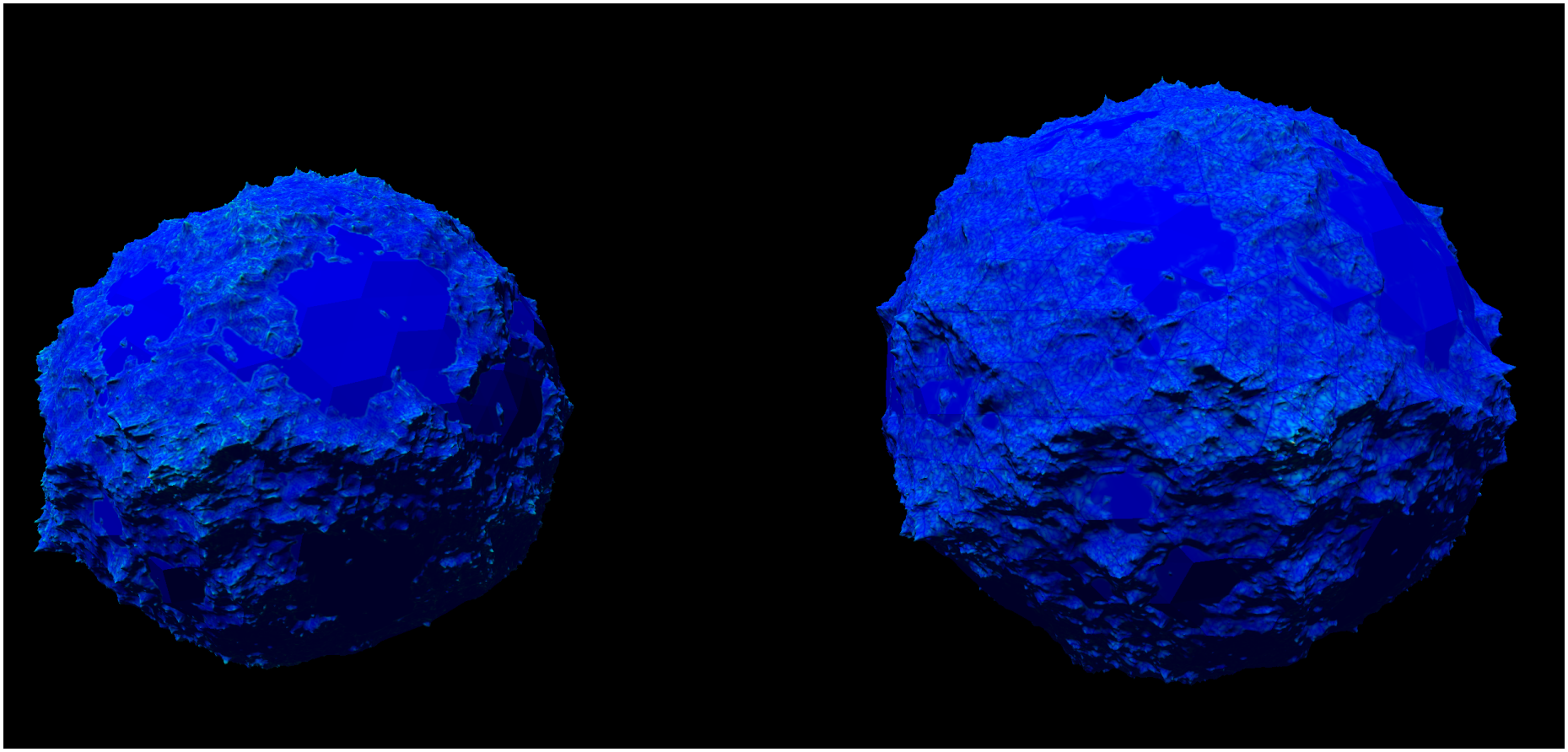

Fig. 28. TGS on the left, QSlim on the right. This shows the geometric deviation heat map for 50 subdivisions at 10% of original geometry. Here QSlim performs slightly better than TGS and that is reflected in the heat map with the TGS planet now having brighter areas but overall, both show a fair amount of deviation

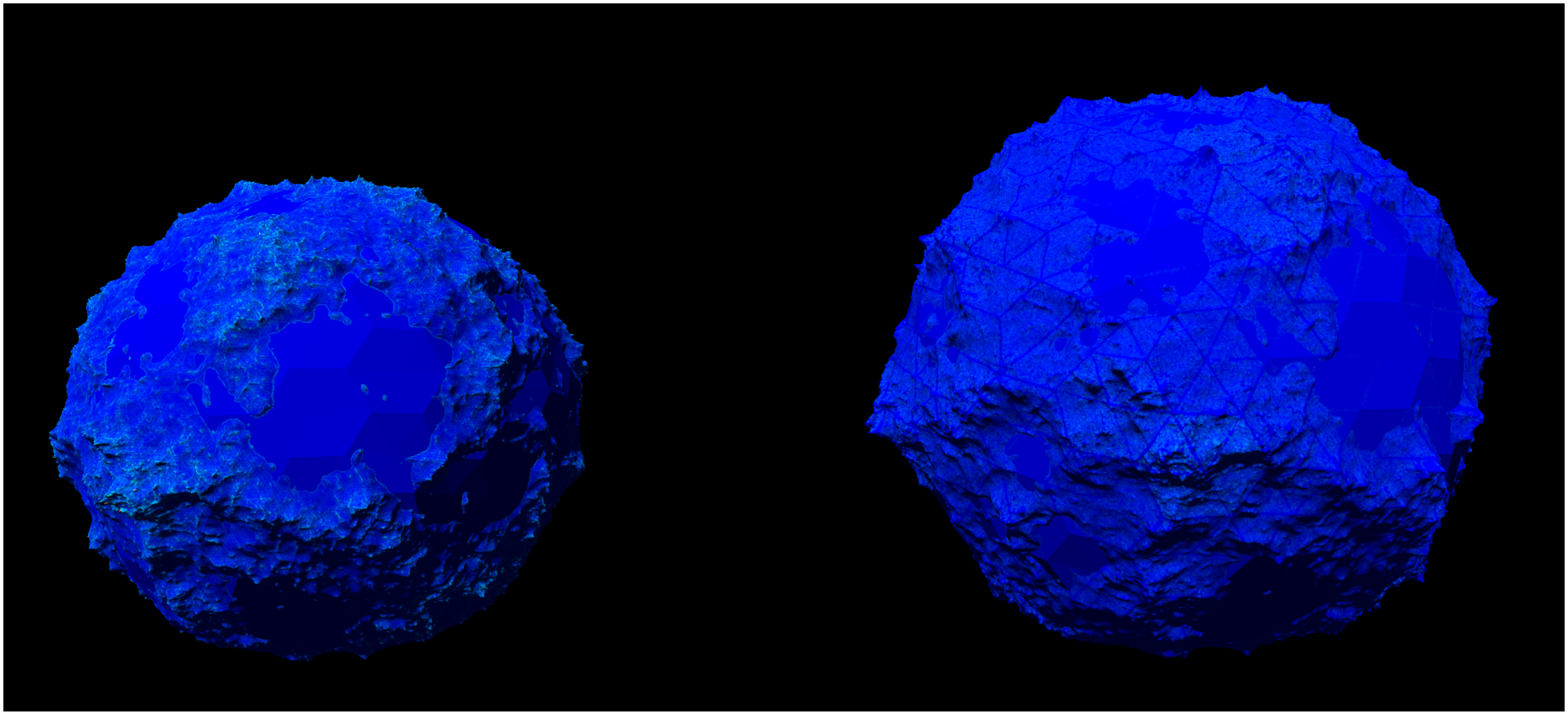

Fig. 29. TGS on the left, QSlim on the right. This shows the geometric deviation heat map for 50 subdivisions at 50% of original geometry. Here QSlim performs much better than TGS and that is reflected in the heat map with the TGS planet now has much brighter areas while QSlin is very dim and almost entirely all dark blue.

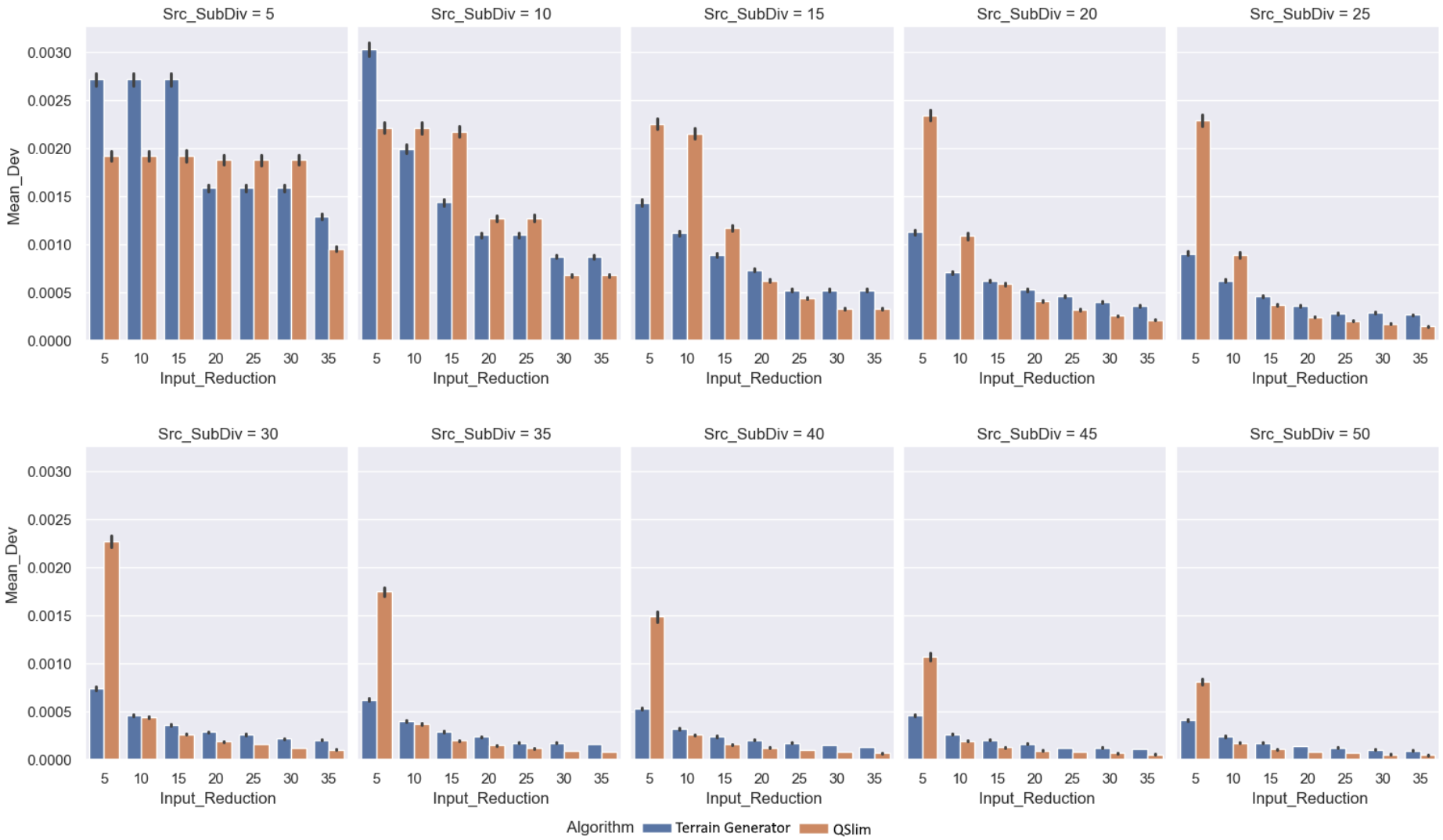

Fig. 30. This graph offers a closer look at the 5% to 35% range at every subdivision level. At 35% QSlim becomes better than TGS at every subdivision level tested.

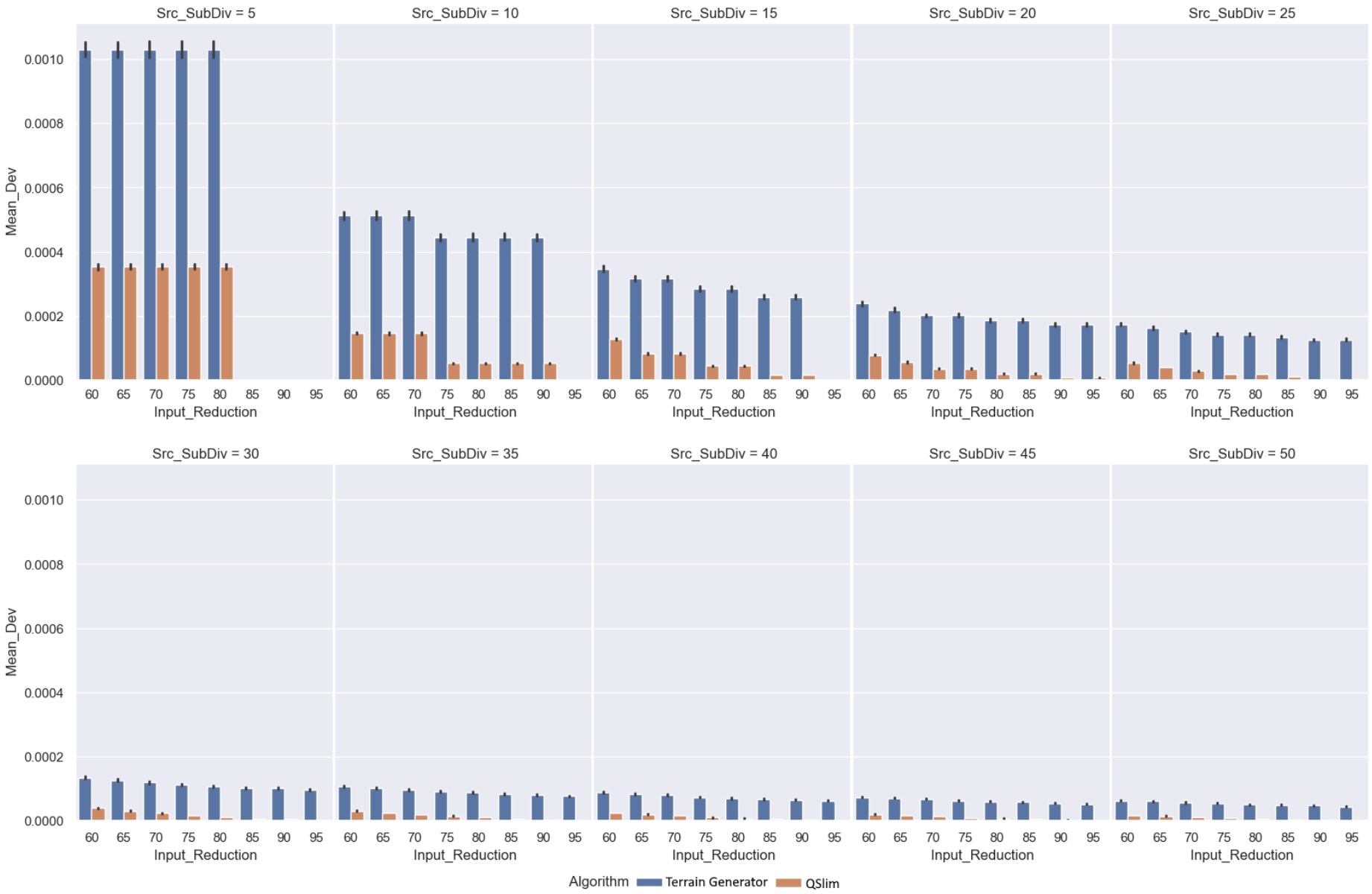

Fig. 31. This graph offers a closer look at the 60% to 95% range at every subdivision level. At this level QSlim is always significantly better than TGS at every subdivision level tested.

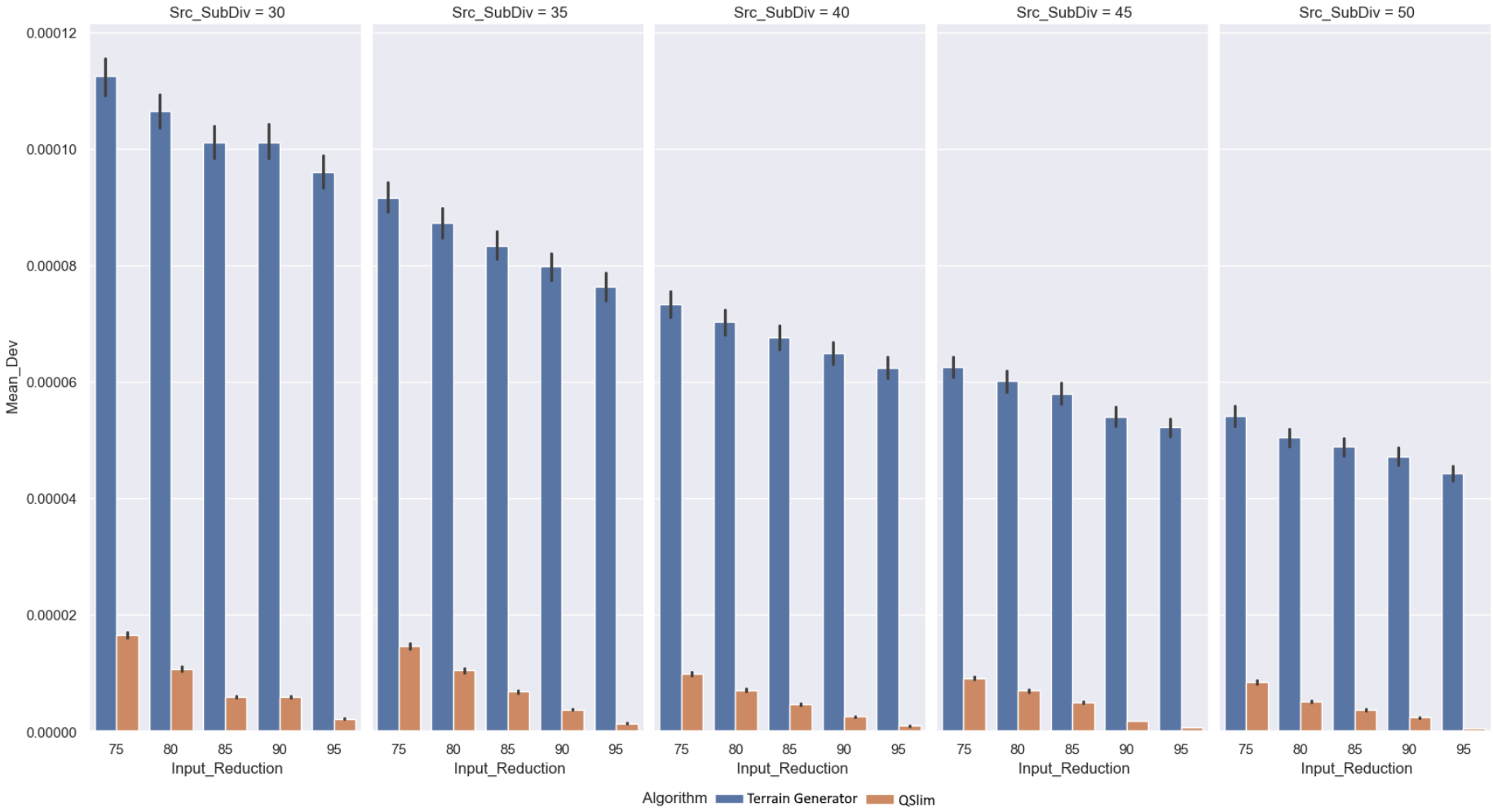

Fig. 32. This graph offers a closer look at the 75% to 95% range at subdivision levels 30–50 so as not to blow the y axis out as much on graphs at 5 subdivisions. At this level QSlim is always significantly better than TGS at every subdivision level tested.

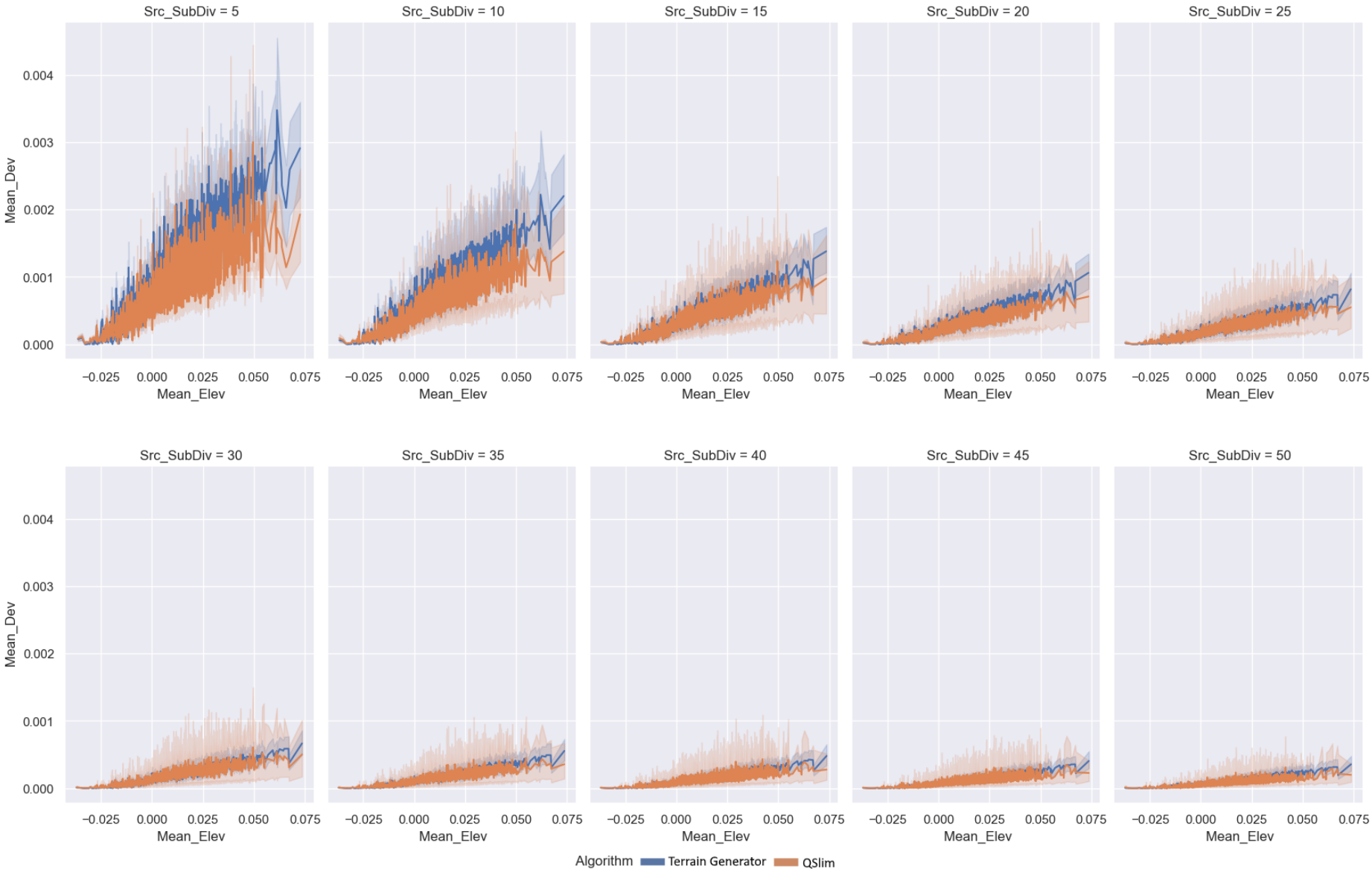

Fig. 33. This graph plots the mean geometric deviation with the mean elevation on every tile. It is expected that in general the greater mean elevation is, the more deviation there would be and this tracks at every subdivision level.

APPENDIX C
TESTING ADDENDUM

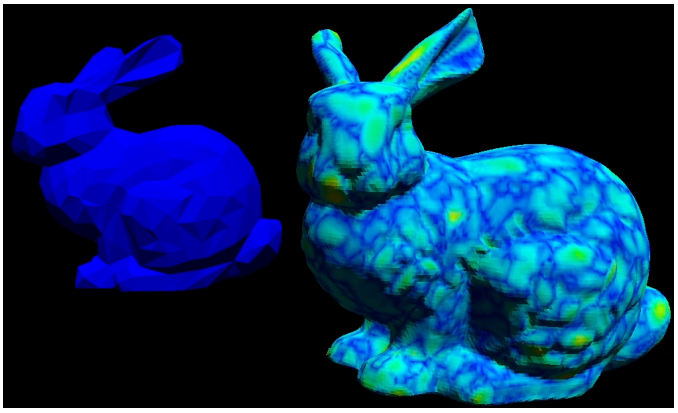

Fig. 34. Test of QSlim and Mean Geometric Deviation showing the Stanford bunny [48] with a heatmap showing areas of difference on the QSlimed blue model.

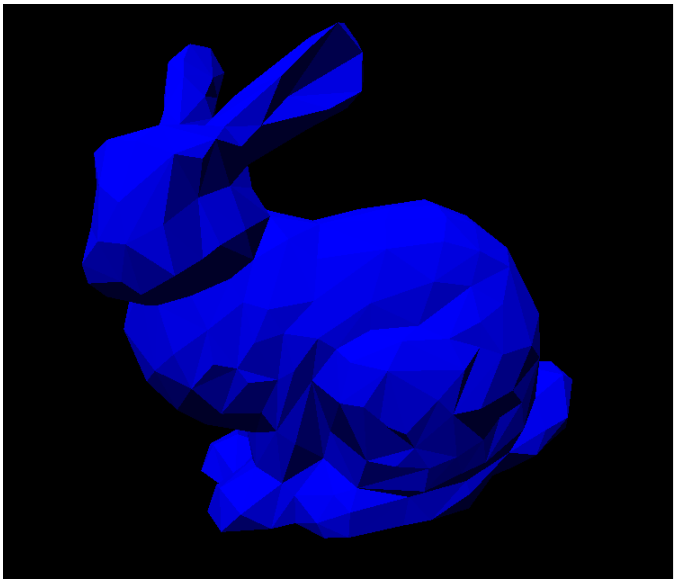

Fig. 35. Close look at the QSlimmed Stanford bunny [48] Results appear very similar to those in the original paper [15]

1) Incidental Stress Testing: Using data gathered from execution time, available from both Test–13 and Test–12 data sets[3] a back of the hand calculation can be made estimating total application run time. The total time logged by the execution time tracking stands at 36673.5s (10.2hrs). This does not include the time to calculate Geometric Deviation, which roughly takes 4.5–15 seconds to run once, each scenario ran it twice for an average time of 70 seconds per scenario and there is 190 scenarios and 10 planets, meaning geometric deviation ran 3800 times for roughly 37 000 seconds per set taking an average of 9.75 seconds per run. Making total geometric deviation time at 74 000 seconds (20.6hrs) combined with the logged time means the application ran in total for 30.8hrs in two 15.4hr sessions. During these sessions no errors or crashes occurred and the data sets were valid apart from non-summary data in Test–12, which is why Test–13 was run, as due to a

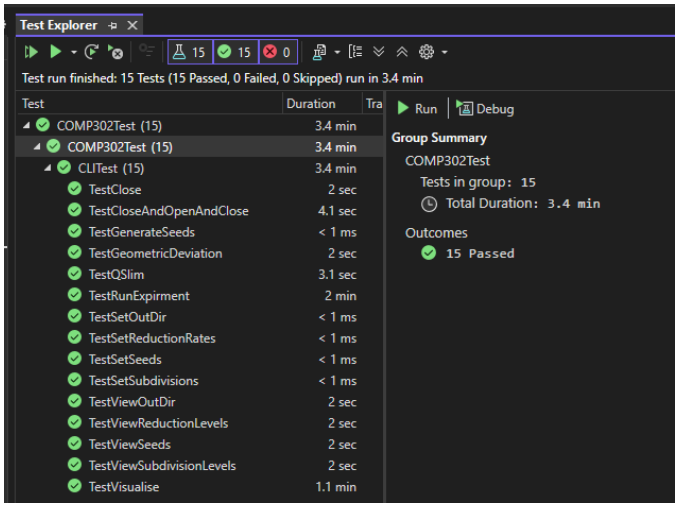

Fig. 36. Evidence of CLI unit testing. Through testing of the CLI core application features also run like QSlim and Geometric Deviation and the Terrain Generator.

syntax logic error, the program exported the same data twice instead of different sets. This indicates the framework at least is very stable and was able run unsupervised for extended periods of time without encountering unexpected behaviour or serious crashes. Note that the software was run on Windows 10 22H2, disconnected from the internet with sleep disabled.

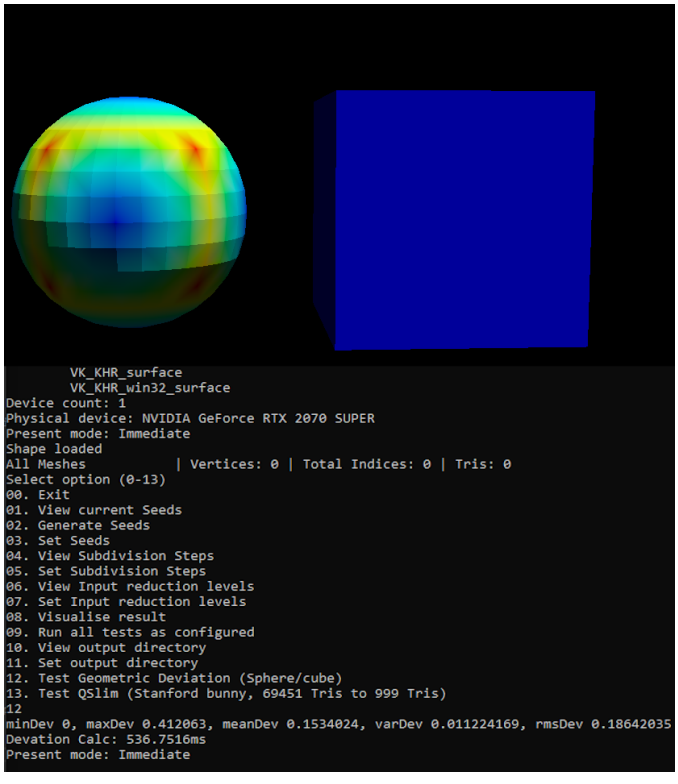

Fig. 37. This is the sphere cube test of mesh dev. This produces a mean geometric deviation result of 0.153 and produces the heatmap on the sphere above highlighting where the corners of the cube are furthest away from the sphere, creating a square yellow shape with red corners. Dark blue areas indicate close areas

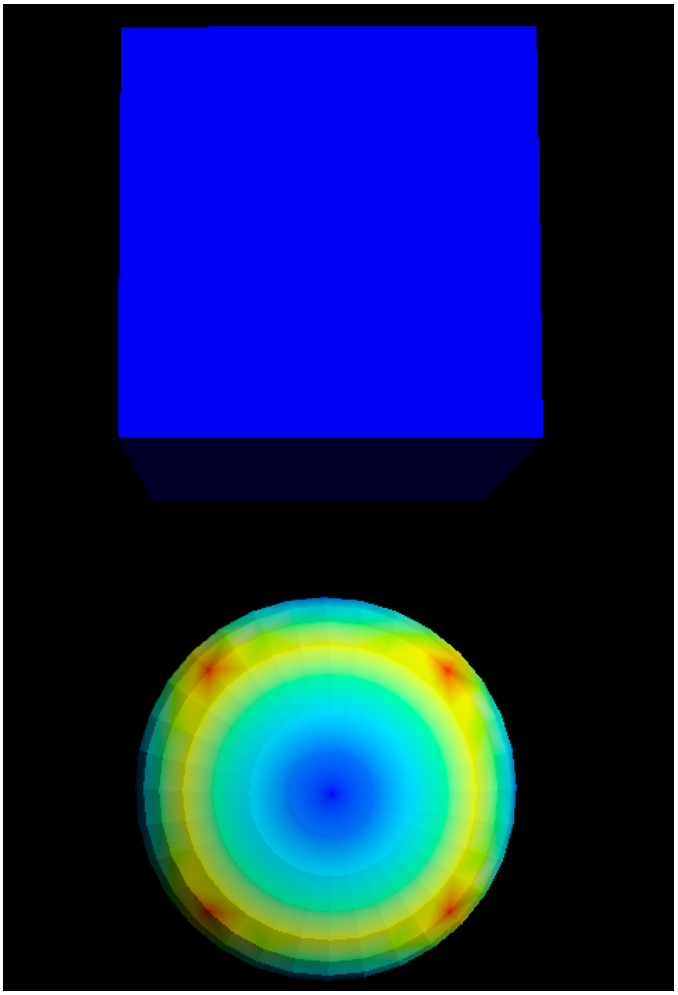

Fig. 38. This figure shows the top down angle of the sphere cube test, where the same pattern in the heat map is reflected. Dark blue in the centre indicating it is close or at the same point as the plane, red corners at the outer edges bordered by a yellow square.

Appendix D
Experiment Operations

Detailed Breakdown of experiment operations

1) Select Subdivision Level, Select Target Reduction Rate.
2) Compute actual reduction rate. This is the actual number of triangles the terrain generator can generate using the subdivision algorithm and defines the actual reduction rate used by the whole scenario.
3) Planet Generation. 4 Planets (A,B,C,D) are initially generated. 3 (A,C,D) are identical and use the current scenario's Subdivision Level. Planet B uses a subdivision level approximately equal to the current scenario's reduction rate, the actual reduction rate as calculated in Step 2.
4) QSlim Simplification. Planet D is simplified using QSlim until has the same number of triangles as Planet B
5) Geometric Deviation (Planet A & B). Mean Geometric Deviation is first calculated for TGS Technique (B) compared against the high–fidelity planet (A). The results are recorded to a dictionary.
6) Geometric Deviation (Planet C & D). Mean Geometric Deviation is calculated for QSlim compared against the high–fidelity planet (C). The results are recoded to a dictionary.
7) Repeat step 1–6 until all subdivision levels and reduction rate levels have been tested
8) Export results. The dictionaries that store the results are converted into a CSV file an exported to a results folder. The application terminates after this.

Appendix E
Reflective Addendum

I am happy with this piece of research; it was a fantastic opportunity to explore mesh simplification and extend my COMP305 module through optimisation of its terrain meshes. This allowed my continued experimentation with game engine development and bleeding edge rendering techniques like GPU driven rendering using Vulkan, even though this was not the main focus of the research.

That said undoubtedly the biggest mistake was a misunderstanding of what Vulkan was capable of specifically with GPGPU Compute This led to my plan of being able to implement Two GPU algorithms by converting them from CUDA compute shaders to Vulkan Compute shaders. This *seemed* reasonable during the proposal but later ended up not being feasible for a variety of reasons that I shall go into.

The other questionable decision was using the custom Vulkan engine for this research. This made testing harder, unit testing was much harder to do, and all the algorithms I implemented I first tried in Unity, so why not just use Unity?

A. CUDA & SPIR–V

At the proposal stage I planned to translate two CUDA compute shader implementations of mesh simplification into Vulkan Compute shaders and then benchmark them on execution speed and shape quality.

For some background, CUDA is a Programming interface developed by Nvidia to enable general purpose GPU (GPGPU) compute through the use of compute shaders, this enables developers to leverage the highly parallelised power of a GPU to accomplish small but very wide tasks. The Vulkan graphics API also supports compute shaders albeit through the Kronos Group's SPIR–V intermediate language.

The issue stems from SPIR–V providing a lot of optional features, which aren't enabled in Vulkan, the two big features not enabled in Vulkan, at least not in a usable way, are memory pointers and kernel memory allocation (being able to allocate memory dynamically during the compute shader's execution), which in practice made translating these compute shaders to Vulkan impossible.

The solution would have been to run a separate CUDA environment or an OpenCL environment and then to parse data between the host application and the GPGPU compute environment to run mesh simplification in, but this would have massively scope up the project.

I didn't do that, luckily I had planned in a contingency which I was able to do instead. What's important here is

how this happened? Simply I just misunderstood how SPIR–V worked, and in fairness to me it wasn't clear that Vulkan *wouldn't* implement the full feature set of SPIR–V. But in hindsight, further research into the feasibility of translating between CUDA and SPIR–V should have been undertaken before I committed to doing so. Additionally, had I started prototyping by trying to implement one of these algorithms instead geometric deviation I would have released a lot sooner than mid–February that this was not feasible.

In the future then, I need to more thoroughly research Frameworks I think I understand of, but don't have experienced with. By the time I committed to compute shaders in the proposal I had been working with Vulkan for a grand total of 2 months. I just assumed it would be easy and work, but I have learnt since that while doing stuff in Vulkan is not hard, it is not easy either and it certainly doesn't work first time. Overall, then going forward I need to strongly consider scope after doing do a large amount of background research into the feasibility of implementation for software artefacts like this in the future.

B. Why not Unity?

For this research I used a Geometric Deviation calculator to measure the difference in shape between two meshes and I used an implementation of QSlim mesh simplification algorithm to generate one of these meshes, to see how effective QSlim was in maintaining shape accuracy.

I first got both of these algorithms working in the Unity game engine, then ported them to run inside my own Vulkan C# framework for the experiment. Why? Why did I do this? This added a lot of extra work, a lot of extra bug fixing and testing needed. The QSlim implementation I chose was even a unity package, why did I use my own framework for this research? And well it boils down to I wanted to.

I think overall using my own framework was a good idea. It enabled me to learn a lot about game engine development in addition to mesh simplification. And why did I attend university? To learn. Doing this module in my own framework I undoubtedly learnt a lot more than I would have just by using Unity, I wasn't on a budget or particularly time restricted. If in the future I am doing something like this on a strict time crunch or on a budget, then Unity would have been a better choice.

in the future then I should always take every opportunity to apply myself beyond what is necessary for the sake of learning and doing something cool, but I need to consider the consequences of doing so in time and stress. If I were to do work like this for another entity, then I should use existing frameworks like unity.

C. Parsing Sudo Code to Executable Code

As part of my initial plans at the proposal stage to translate CUDA compute shaders to Vulkan ones, I needed to gain a deep understanding of the algorithms I would be implementing. Neither of the algorithms provided source code, only pseudocode in their papers, a more minor reason for abandoning my proposal plan was I couldn't parse this pseudocode into executable code, I really struggled to understand, and I think part of that was some of the pseudocode was nonsensical, but mostly it is my lack of understanding it.

I need to learn to read and parse pseudocode. I know already I have high comprehension of C# and C++ I can reverse engineer undocumented code easily, its very interesting that I struggle with this more abstract way of communicating code in a standard way. I believe this is because I have for most of my time as a programmer worked alone on very large code bases. There hasn't been a need for other people to understand my code, this is not a good practice as being able to have your work understood by peers is very crucial in the software industry working in teams to deliver games or other products.

D. Successful Contingency Planning

As previously mentioned, I had to pivot my research from comparing two simplification algorithms to each other to the topic I actually did which was a wide–ranging analysis of QSlim on convex terrain. I believe this was a successful pivot; my results and findings provide an interesting look at how QSlim behaviours in different scenarios in an area without really any prior work. Why was the pivot successful though? The answer is I had a contingency plan, going into the proposal I wanted to test mesh simplification on generated terrain. I already had the terrain generator; I was missing the simplification and a way to measure it and I wanted to shoot for compute shader based simplification was it is the bleeding edge and it seemed plausible to do.

I knew it was only plausible not definite and the actual value in the research I was conducting was an investigation of mesh simplification on convex terrain generation, what simplification algorithm wasn't very important as long as could use *an* algorithm, I knew from research into the area a large number of tools implemented QSlim, and that QSlim had source code released. As a relatively old piece of software now coming from 1997, it is very well understood and well used and available for Unity as package.

This meant my worst–case scenario was testing QSlim on generated terrain in Unity because I had all the building blocks prior to doing anything beyond reading. Luckily it wasn't the worst case as I was able to use my own artefact but knowing I could do this gave me the confidence to push for the bleeding edge compute shader algorithms knowing I could fall back on this simpler idea.

I am quite a risk averse person, so I find this form of planning quite natural it's definitely a good skill to maintain into the future when planning higher scope projects to build in contingencies from day 1 so scope can be reduced if required and also understanding what you are capable of, so you don't over-scope in the first place.

E. SMART Objectives and Conclusion

1) Planning and Research: Not-so-SMART objective 1: I need to do more background research into the feasibility of projects before I commit to them.
Specific: Greater depth of research into feasibility of projects.
Measurable: Having to down scope in future projects should

become less frequent.
Achievable: Taking extra time to plan and research is very achievable, it's finding the motivation to that's challenging.
Relevant: This goal relates is valuable in professional practice so I can give accurate estimations of what I can do and how long it will take
Time-Bound: I have plans to do some projects after the dissertation and over the summer. From Easter break until August.
SMART objective 1: Between Easter and August, put aside time at the start of projects to thoroughly research and understand the requirements needed to fulfil them so adjustments to the project scope prior to starting can be made, which should be visible through lowered scope reductions being needed or an increase in projects completed.

2) Always Push to Learn: Not-so-SMART objective 2: I want to continue to take every chance to go the extra mile with work for the sake of learning and doing something cool.
Specific: I am going to continue learning GPU Driven rendering techniques as it's something I started but had to put on hold to complete my dissertation, it's an area I want to learn about
Measurable: I want to get depth-based occlusion culling working in a Vulkan compute shader, progress towards that and understanding it is how I shall measure this for now.
Achievable: I have an understanding of the theory and some ideas of how to implement it. Currently I feel confident I can.
Relevant: I'm hoping to use this as a portfolio piece to help me get hired to work as a graphics programmer or engine programmer.
Time-Bound: I want to get back into this project as soon as the dissertation is over, and I want to have it complete by the expo (may 21st 2025).
SMART objective 2: From now until May 21st I want to push myself to implement advanced GPU driven rendering techniques to support my portfolio and demonstrate my ability to push myself to learn bleeding edge topics.

3) Planning to Avert Failure: Not-so-SMART objective 3: I want to ensure contingency plans in my future research so that in case of unforeseen issues valuable knowledge is still gained.
Specific: In future research work, plan for cases where what is hoped isn't possible and to still extract useful work from that.
Measurable: This is dependent on if I do more academic R&D projects in the future.
Achievable: I have demonstrated in this piece of work that its possible, so it is feasible for future research projects I undertake.
Relevant: Being able to plan for failure is useful for mitigating its effect in life and in jobs, something at some point will go wrong and being able to respond effectively to it is invaluable.
Time-Bound: This is definitely a far future task contingent on another academic research project. Outside of that it's something I find myself doing every day.
SMART objective 3: Future academic research projects I undertake I want to plan in a similar way to this project where if I am unable to meet the desired goals, I can still produce a piece of valuable work of high quality and of some value.

4) Conclusion: Although things did not pan out hope I originally hoped and planned, I feel this piece of work is worth being proud of. Even if it is not marked again here the underlying graphics engine performed well again and was a stable framework which enabled me to collect good quality data. This is my second major piece of research since a physics project I undertook in 6th Form and it's on a far higher scale and I believe far higher quality than that was. This project taught me about advanced statistical tests, something I had almost no experience with beforehand and through development has provided me with valuable skills and knowledge within my specialism and also with academic writing. In hindsight I should have done more background research but, this had a relatively minor impact overall. I don't think I would do this project differently from the way I have, and I am pleased with how it has turned out. This reflection has provided me with some directions to go regarding future projects of my own and I shall continue to develop my skills in aide of that.

APPENDIX F
DATA ANALYSIS R CODE

```
# The computing arefact provides several CSV files that can be used for data analysis.
# Two csv files are intented to be compared together, one csv file contains the data
# for the terrain generation algorithm and the other for the quadric simplification
# algorithm All data is output twice, the difference is the order of the rows within
# the set, one is ordered by subDivisions then input reduction rate, the other by
# input reduction rate then by subDivisions
# Files that start with SUMMARY contain an the mean data for a whole planet.
# Files that start with EXE contain only the execution time for
# the generation/simplification operation for a whole planet. These files have not
# been analysed as execution performance is not within the scope of this paper
#
# For this data anylsis the full data sets are used.
# These data sets record results for each tile mesh that composes a planet.
# The seed shows what planet any given tile belongs to.
# the Tile index identifies a tile within a planet.

### Independent variables ###
# Algorithm (0 or 1) identifies what algorithm was used to produce a simplified mesh.
# Algorithm 0: Terrain Generator
# Algorithm 1: Quadric Simplification (QSlim)
# Src_SubDiv: The number of additional vertices added to each edge of each triangle
# of the original mesh. - How much geometry is in the mesh to start with.
# High values = more geometry

# Input_Reduction:
#   indicates the requested whole number percentage of original geometry to keep.
#       95 > keep 95% of original geometry - reduce mesh by 5%
#       05 > keep 5% of original geometry - reduce mesh by 95%
#       It is important to note this is a REQUESTED input.
# The actual number of vertices and triangles in the final mesh may not
# match these values because a vertex/triangle doesn't divide smaller than 1

### Dependant varaibles ###
# Vert_Count: Number of vertices in the simplified mesh
# Tri_Count: Number of triangles in the simplified mesh
# Vert_Reduction: % of vertices left compared to original mesh
# Tri_Reduction: % of triangles left compared to original mesh

## Elevation - defines how far in the local y axis of the mesh
## each vertex gets displaced to produce mountains/hills/flat ground ##
## Effectively these values indicate how much the mesh has been displaced,
## values towards 0 = low displacement ##
# Min_Elev: Lowest point of vertex displacement in the final mesh
# Max_Elev: Highest point of vertex displacement in the final mesh
# Mean_Elev: Aveage point of vertex displacement in the final mesh

## Shape quality ##
# Geometric devation from original mesh,
#if min, max and mean = 0 there is no difference between the start and end mesh

## Overall quality is measured by mean geometric devation (Mean_Dev) ##
# Min_Dev: Smallest geometric devation from the original mesh
# Max_Dev: Largest geometric devation from the original mesh
# Mean_Dev: Mean geometric devation from the original mesh
```

```
### Data source files ###
# SUB_Terrain_Generator.csv
#     Full Terrain generation quality set ordered by subDivisions
# SUB_Quadric_Simplification.csv
#     Full Quadric simplification quality set ordered by subDivisions
# These are compbined into one data set, combined_Set

library(readr)

SUB_Terrain_Generator <- read_csv(
"H:/GitFork/Comp-302-2202796/Data/Test-13/SUB_Terrain_Generator.csv",
    col_types = cols(Seed = col_integer(),
        Tile_ID = col_integer(),
        Algorithm = col_factor(levels = c("0", "1")), Src_SubDiv = col_integer(),
        Input_Reduction = col_integer(),
        Vert_Count = col_integer(), Tri_Count = col_integer()))

SUB_Quadric_Simplification <- read_csv(
"H:/GitFork/Comp-302-2202796/Data/Test-13/SUB_Quadric_Simplification.csv",
    col_types = cols(Seed = col_integer(),
        Tile_ID = col_integer(),
        Algorithm = col_factor(levels = c("0", "1")), Src_SubDiv = col_integer(),
        Input_Reduction = col_integer(),
        Vert_Count = col_integer(), Tri_Count = col_integer()))

combined_Set <- rbind(SUB_Terrain_Generator, SUB_Quadric_Simplification)

### Data anylsis ###
# We are looking at how The algorithm used, Src_SubDiv and Input_Reduction
# effect overall quality (Mean_Dev).
# Multiple Statistical tests where run to analyse the data set.
# First, a T-test comparing Algorithm to Mean_Dev on the whole data set is run to
# validate there is a difference between sets.
# Second, cohen's d is calculated for Algorithm and Mean_Dev across the whole data set
# to calculate the overall effect size.
# Thirdly a linear model regression test is run to see how the three factors
# Algorithm, Src_SubDiv and Input_Reduction effect Mean_Dev.
# Fourth, a pratt analysis of the model is run to calculate the importance of
# each regressor.
#
# Finally, a t test is conducted for each expirmental test run.
# The focus of this test is to see how Mean_Dev varies isolating to
# only the Algorithm.
# The Difference between the two data sets being compared here is the
# algorithm being used, Src_SubDiv & Input_Reduction are constant for each t.test.
# This is done to test if the Algorithm difference is universally better and
# to highlight anomalies in the data set.
# Cohen's D is also calculated for each test run.
#
# Additional notes.
# - The sink command is used to record the data anylsis to a text file.
# - Some t tests from each expirmental test section show NAN t values but still
#   state the alternative hypthsis is true.
#       This is incorrect. In such cases the alternative hypthsis is false,
#   there is not difference in Mean_Dev between the sets.
#   The mean in group 0 & mean in group 1 right below this statement
#   show that in all cases.
```

```
library(rstatix)
library(relaimpo)
sink("Data_Analysis.txt")
print("Overall_T-Test")
statistic <- t.test(Mean_Dev~Algorithm, data = combined_Set, Paired=TRUE)
print(statistic)
print("Overall_Cohen's_D")
cohens_d <-combined_Set %>% cohens_d(Mean_Dev~Algorithm, paired=TRUE)
print(cohens_d)
print("")
print("")


print("Linear_Model")
lm_results <- lm(Mean_Dev~Algorithm+Src_SubDiv+Input_Reduction,data = combined_Set)
summary(lm_results)
print("Pratt_analysis_of_Linear_Model
-product_of_the_standardized_coefficient_and_the_correlation")
lm_pratt <- calc.relimp(lm_results,type=c("pratt"),rela = TRUE)
print(lm_pratt)

print("")
print("")
print("Per_expirment_T-tests")

tTestResults <- vector("list",190)
cohens_dResults <- vector("list",190)
counter <- 1
tTestResults <- list()
for(s in 1:10)
{
    for(i in 1:19){
                statistic <- t.test(
                            Mean_Dev~Algorithm,
                            data = combined_Set[
                                combined_Set$Src_SubDiv == subDivisions[s]
                                & combined_Set$Input_Reduction == inputReductions[i],
                            ], Paired=TRUE)

                cohens_d <-combined_Set[
                            combined_Set$Src_SubDiv == subDivisions[s]
                            & combined_Set$Input_Reduction == inputReductions[i],
                            ] %>% cohens_d(Mean_Dev~Algorithm, paired=TRUE)

                statistics[[counter]] <- statistic
                cohens_dResults[[counter]] <- cohens_d

                identifier <- toString(
                                c("Expirment-SubDivisions:_",
                                subDivisions[s],
                                "InputReductions:_",
                                inputReductions[i]))


                print(identifier)
                print("t-test")
                print(statistic)
```

```
                print("Cohen's_D")
                print(cohens_d)
                print("")
                print("")

        counter <- counter + 1
    }
}

sink(NULL)
```