Bachelor in Digital Game Development
June 2018

Image Quality-Driven Level of Detail Selection on a Triangle Budget

Ludvig Arlebrink
Fredrik Linde

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Bachelor in Digital Game Development. The thesis is equivalent to 10 weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:
Author(s):
Ludvig Arlebrink
E-mail: luar15@student.bth.se

Fredrik Linde
E-mail: frld15@student.bth.se

University advisor:
Universitetsadjunkt Francisco Lopez Luro
Department of Creative Technologies

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Background. Level of detail is an optimization technique used by several modern games. The level of detail systems uses simplified triangular meshes to determine the optimal combinations of 3D-models to use in order to meet a user-defined criterion for achieving fast performance. Prior work has also pre-computed level of detail settings to only apply the most optimal settings for any given view in a 3D scene.

Objectives. The aim of this thesis is to determine the difference in image quality between a custom level of detail pre-preprocessing approach proposed in this paper, and the level of detail system built in the game engine Unity. This is investigated by implementing a framework in Unity for the proposed level of detail pre-preprocessing approach in this paper and designing representative test scenes to collect all data samples. Once the data is collected, the image quality produced by the proposed level of detail pre-preprocessing approach is compared to Unity's existing level of detail approach using perceptual-based metrics.

Methods. The method used is an experiment. Unity's method was chosen because of the popularity of the engine, and it was decided to implement the proposed level of detail pre-preprocessing approach also in Unity to have the most fair comparison with Unity's implementation. The two approaches will only differ in how the level of detail is selected, the rest of the rendering pipeline will be exactly the same.

Results. The pre-preprocessing time ranged between 13 to 30 hours. The results showed only a small difference in image quality between the two approaches, Unity's built-in system provides a better overall image quality in two out of three test scenes.

Conclusions. Due to the pre-processing time and no overall improvement, it was concluded that the proposed level of detail pre-preprocessing approach is not feasible.

Keywords: image-quality, triangle budget, level of detail, simplification

Acknowledgments

We would like to thank our supervisor Francisco Lopez Luro for his feedback and guidance throughout the project.

Contents

Abstract **i**

Acknowledgments **iii**

1 Introduction **1**
1.1 Motivation . . . 3
1.2 Research Question . . . 3

2 Related Work **5**
2.1 Rendering on a power budget . . . 5
2.2 Screen-space error constrained simplification . . . 5
2.3 DLOD approximations in a hybrid framework . . . 7
2.4 Usage of perceptual difference models . . . 7

3 Method **9**
3.1 Framework Implementation . . . 9
3.1.1 Stage 1 . . . 9
3.1.2 Stage 2 . . . 10
3.1.3 Stage 3 . . . 11
3.2 Experiment Design . . . 11
3.2.1 Scenes . . . 11
3.3 Data Gathering and Measurements . . . 12
3.4 Experimental Setup . . . 15

4 Results **17**
4.1 Pre-Processing . . . 17
4.2 Runtime Tests . . . 18
4.2.1 Runtime Test Scenes Data . . . 18

5 Analysis and Discussion **29**
5.1 Analysis Scene 1 . . . 30
5.2 Analysis Scene 2 . . . 32
5.3 Analysis Scene 3 . . . 34

6 Conclusions and Future Work **37**
6.1 Future Work . . . 37

References **39**

Chapter 1

Introduction

Level of detail (LOD) is an optimization technique used to render the appropriate geometrical complexity representation of a triangular mesh, depending on its contribution to the rendered scene [22, p. 12]. The philosophy behind LOD is that from a distance the user will not be able to tell the difference between a more or less complex triangular mesh [22, p. 5]. The most common LOD technique is discrete LOD (DLOD), which uses several geometrical representations of a triangular mesh of different qualities.

DLOD is often used with transition thresholds to switch between different polygon count versions [14, 27]. The most common ways of determining the contribution to the rendered scene are by *ranged-based* and *projected area-based* [22, p. 232] [6, p. 688]. Range-based uses a distance criterion to determine which LOD to use based on the distance to the camera; Projected area-based uses the projected size of the mesh on the screen to determine which LOD to use. The higher the size of an object on the screen, the higher the level of detail to be used.

In Unity's projected area-based solution, this happens in response to configured thresholds regulating the amount of space that the object currently occupies on the screen [4]. To avoid errors from manually configured thresholds, a recent implementation by DICE [11] uses a pre-computing stage to store LOD versions for real-time applications in Frostbite engine. The algorithm approximates transition thresholds for LOD versions to minimize artifacts in the triangular meshes silhouettes when changing between the LOD versions. Since it is primarily focused on reducing artifacts of silhouette changes, texture deviations and light artifacts were still introduced during LOD transitions [11, p. 33]. Similarly to approximating LOD thresholds in a pre-computing stage to minimize image artifacts from triangular meshes, it should be possible to approximate LOD combinations that minimize the loss of quality in an image.

This paper proposes an image quality-driven process to choose a LOD combination given a camera position and orientation, and a triangle budget. The metric to assess the quality of the rendered image is the *structural similarity* (SSIM) index, which is a popular choice in computer science for image comparison for its ability to approximate similarity between images [32, 24, 12]. However, since SSIM is only an approximation of similarity, it cannot identify the exact pixels in a region where the changes occur in an image. A simplified *perceptual difference model* (PDM) in

a perceptual image comparison utility based on Daly's *visible differences predictor* (VDP) will be used to locate these structural differences. VDP does not describe an absolute value of image quality but instead addresses the problem of differences between images [33, 8]. This yields an additional metric of perceptual pixel difference.

To be able to implement this in real-time, a LOD combination for all meshes will be pre-computed for each camera from a finite set of representative positions, and without exceeding a total triangle budget. This exploratory step, will pre-render from each reference camera position a combination of LOD meshes, and then compare it using the SSIM against the reference image. This reference image is produced using the highest available mesh qualities. The LOD combination is determined by only lowering the meshes with least impact on SSIM for each reference camera and will be stored for real-time use. As this idea is also driven by a triangle-budget, LOD combinations that exceed the budget are not included. As the camera in real-time can be in any place, the closest reference camera position and angle will be used to determine the LOD combination to use as seen in figure 1.1. The pre-render camera positions are laid out in a structured manner following a sparse quadtree approach, that can be further subdivided only when more reference camera positions are needed [15].

This approach is expected to produce a similar SSIM index compared to the reference quality when the camera position and angle are close to a reference camera position and angle. However, there are limitations to consider. The pre-rendered camera positions have to be representative of where the real-time camera can be. For this, a first-person style camera will be used to fly across the open scene where the pre-render cameras have been used. It is possible that more pre-render positions are needed in scenes with many meshes. The process will also only work with static meshes, as dynamic objects can affect significantly the choice of a LOD combination for a given triangle budget. Because the approach is not trying all possible LOD combinations, it cannot be ensured that it will find the best possible LOD combination in terms of SSIM. Since the proposed approach has not been narrowed down to only use ambient light, it is possible that image artifacts can be caused by lighting deviations.

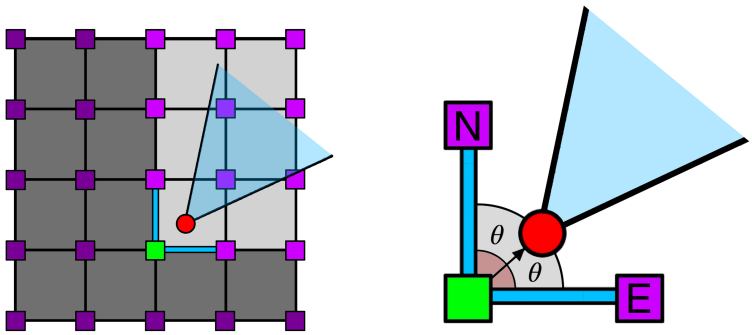


Figure 1.1: Overview of the image-quality driven LOD selection approach. The main camera position and angle are close to a reference camera position and angle.

1.1 Motivation

Switching thresholds are for example set up by an artist, by "eyeballing" switching parameters. The ranged-based or the projected area-based solution only works on an object basis and does not take the scene into account. For example, objects that are half or fully occluded, but remain relatively close to the camera might not need high complexity. This could also include low-lit areas, where objects are not lit.

However, before testing the image quality-driven approach on the scene with varied lighting, it is important to see if this approach is viable in general. By viable in this context, it means that is close or on a par to the projected area based solution that Unity uses.

Despite that the proposed approach is primarily developed for the personal computer platform (PC), it is more likely to contribute the most to rendering complex scenes on mobile devices. This is because large productions rarely have to be concerned with triangle budgets using the modern day hardware, but mobile devices are running on stricter triangle budgets due to their limited hardware in comparison with the PC. Even so, image quality must still be preserved and maintain a high quality on the triangular meshes even if a lower triangle count is required. That is where the proposed approach differs from the previous DLOD approaches by also taking image quality into account, and not only performance.

1.2 Research Question

Unity's default approach for handling *level of detail* is based on *projected area* and relies on a user-defined threshold for the level of detail *switching*. The proposed approach could potentially lead to similar SSIM quality compared to Unity's built-in approach to LOD. Therefore, this paper is concerned with the following research question:

How does the SSIM quality of rendered images, from a free moving camera, compare between Unity's built-in approach to LOD and the proposed pre-computed approach?

Chapter 2

Related Work

Similarly to optimization techniques, pre-computation of optimal render settings and perceptually driven simplification for triangle budgets have previously been studied in the past. The previous research presented in the following sections is intended to introduce the concepts borrowed for the purpose of this research.

2.1 Rendering on a power budget

A recent approach showed an application created for the purpose of rendering on a power budget for mobile devices [31]. The framework used in the application was based on an octree data structure where each corner saved pre-computed rendering settings in six cameras forming a view cube. By finding the frustum with the closest angle to the main camera, only the most essential and optimal rendering settings would be applied. By discarding irrelevant rendering settings for the given view, battery lifetime was significantly improved. For image comparison, the authors used SSIM to compare the images produced by their pre-computed rendering settings with the lowest and highest rendering settings available for each scene.

2.2 Screen-space error constrained simplification

Luebke et al. [20] have previously studied perceptually driven rendering by implementing a LOD framework using a triangle budget. This was to allow real-time rendering of complex CAD models by using the framework for dynamic view-dependent simplification. The view-dependent simplification algorithm relied on three criteria: a triangle budget, a screen-space error threshold, and a silhouette test. By combining the triangle budget with perceptually driven-metrics, this study showed how pre-processing algorithms and runtime criteria for LOD could be used to render complex scenes in real-time.

The screen-space error threshold allowed the user to set a limit on the image quality of the simplified scene, which in this case is the allowed amount of fidelity to be visible in the rendered image. Fidelity is the ability to discriminate changes between two images and is commonly caused by noise introduced from a lower pixel precision where not all detail is considered [28]. Often a limit on the geometrical complexity of a 3D model is also desired. Instead of representing the entire scene as a collection of objects each rendered at different levels of detail, the *hierarchical dynamic simpli-*

fication (HDS) algorithm in the framework used a vertex tree. That is, a hierarchy of vertices which is queried dynamically to generate a simplified scene [26, p.747].

The underlying philosophy of HDS is to both remove triangles that are not important to the scene, and to minimize the maximum screen-space error of all nodes within this triangle budget. The triangle budget drives the simplification to allow the user to specify how many triangles the scene should contain. The screen-space error is then introduced by collapsing the vertices. This means that polygon vertices are moved from their original position and collapsed together into a single vertex. Polygonal detail is removed from the mesh when the vertices are collapsed and this results in a coarser geometry. From this error, HDS establishes a quality constraint on the simplification by not allowing vertices to move by more than a limited number of pixels on the screen. The proposed approach uses a similar quality constraint by only lowering the LOD for meshes with least impact on SSIM.

Future work to the framework saw the introduction of additional perceptual-based metrics to evaluate simplification based on the Contrast Sensitivity Function (CSF) and spatial frequency in an image [21]. Since these are two of the most important metrics commonly used in a PDM implementation, it is not surprising that both of these are part of the simplified PDM used in the perceptual image comparison utility. In order to understand its importance for image comparison, contrast sensitivity itself is the ability to perceive changes in luminance between areas that are not separated by distinct borders. CSF is a measurement of humans ability to detect such a low contrast pattern often appearing as vertical stripes of shades falling steadily from black to grey. In turn, CSF depends on the spatial frequency which determines how rapidly changes in patterns takes place across an image [24, 12].

The study presented a simplification approach to displaying the best possible image quality within given polygon constraints based on a "worst-case" contrast and frequency introduced by the operations. Worst-case, in this case, refers to the simplification operations with the most perceivable combination of image artifacts concerning contrast and frequency possibly induced by performing the operation. Therefore, only the simplification operations judged imperceptible were applied and the resulting simplified model was indistinguishable from the original by also emphasizing on silhouette preservation. However, manifold topology is not always assumed nor preserved since these frameworks are more suited for rendering complex hand-crafted CAD models. Manifold topology has a configuration so that a triangular mesh can be split along its edges and unfolded so that the geometry lays flat without overlapping faces. In contrast to manifold geometry, a non-manifold triangular mesh have problematic configurations, such as invalid edge connections, that makes it impossible for the geometry to be unfolded in a continuous flat space [16]. However, assuming a manifold geometry greatly reduces the options for simplification. As pointed out by Cohen et al. at the time, it is generally easier to identify non-manifold topology than repairing it [10].

2.3 DLOD approximations in a hybrid framework

Hilbert and Brunett based their hybrid LOD rendering solution on the previous approach by integrating geometry and image-based LOD mechanisms in one system. A pre-computing stage was used to generate a continuous multi-resolution model for all objects. In runtime, each object could be checked to find out if an appropriate cached approximation generated for previous frames was available. Knowledge of the current viewing conditions was necessary in order to extract the correct adaptive-geometry based approximation of an object out of its vertex tree. The list of active triangles corresponds to a "cut" that is moved through the vertex tree determined by a view-dependent criterion. The criterion is based on the same screen-space error used by Luebke.

If reasonable and efficient, the geometry based approximation was replaced by an impostor to drastically reduce the number of polygons [17]. By caching the image-based or geometry-based approximations of LOD objects in an approximation cache, the approximations could be reused for several frames. Despite being a view-dependent solution, the process of approximating and caching the optimal DLOD combinations for a given camera frustum to meet a triangle budget is similar to the image quality-driven selection. Also, the DLOD combinations can be reused at runtime.

2.4 Usage of perceptual difference models

Sundstedt et al. [29] used Daly's VDP in their study that the simplified PDM is based upon. Their work was exploring if an image can be selectively rendered when a user is performing a visual task in an environment. The fundamental purpose of perceptual difference models is to mimic the known parameters of the human visual system (HVS). In driving simulations, the user is most likely looking for street signs and the HVS can be exploited to save rendering time by not computing a high quality of those parts of a scene that it will fail to notice. Quality parameters such as image resolutions, edge anti-aliasing, reflections, and shadows were altered to investigate to what level viewers fail to notice any degradations in image quality. However, the simplified PDM used for the proposed approach is simplified in comparison to other PDM implementations used for health care. While there are many variations of PDM, Case-PDM has been shown to be superior to other automatic evaluation metrics, especially for the evaluation of medical imaging [24].

Case-PDM features additional perceptual metrics such as edge detection which is not present in the image comparison utility. It should be noted that these metrics are only used to find out if a pixel qualifies as perceptually different. If multiple types of artifacts were required to be measured separately, it would instead be appropriate to follow the implementation of Artifact-PDM which is a more developed version of Case-PDM [23]. Most perceptual difference models average only a single scalar image quality metric, such as the simplified PDM implementation with its perceptually different pixels. This missing functionality and the absence of additional metrics makes it simplified in comparison to the Case-PDM and Artifact-PDM.

Chapter 3

Method

The scientific method is an experiment, where an alternative LOD selection technique will be implemented and compared to Unity's built-in method. Unity's method was chosen because of the popularity of the engine, and it was decided to implement the proposed approach also in Unity to have the most fair comparison with their implementation. The two approaches will only differ in how the LOD versions are selected, the rest of the rendering pipeline will be exactly the same. To compare Unity's solution to the proposed approach, the thresholds must be similar to the proposed framework. Regardless of the thresholds selected, these will define a triangle budget in Unity, that the proposed approach will be also limited to.

3.1 Framework Implementation

The framework for the proposed approach consists of three stages: generation of LOD meshes for each mesh, a pre-computing step to determine view-dependent LOD versions, and a real-time stage where these combinations are used in the game based on the camera position.

3.1.1 Stage 1

The first stage generates five new meshes with fewer triangles. Examples of mesh resolutions provided by established companies from the game industry were used as a reference to guide the complexity of the original meshes. As demonstrated by Quantic Dream at an E3 presentation in 2013, the mesh resolution for the main character in their PlayStation 4 title *Beyond: Two Souls* was 30000 triangles [9]. In the following year, Guerilla Games held a session at the Game Developers Conference about their PlayStation 4 title *Killzone: Shadow Fall* where the mesh resolution per character was 40000 triangles [30].

The algorithm to create the LOD versions is the *incremental decimation* algorithm [25]. For more implementation details of incremental decimation, see the works by Botsch et al [7, p.115]. It is based on the *Fast-Quadric-Mesh-Simplification* project [1]. The same project was rewritten for C# and it has the same functionalities of the original project, with the exception of being completely adjusted for usage in Unity. This rewritten version of the project is called *Unity Mesh Simplifier* [5]. For

more detail on quadric errors used in incremental decimation, Garland and Heckbert [13] were the first to use Quadric Error Metrics as the criteria for pair contraction of vertices for surface simplification.

Another approach would be to use *vertex clustering* which is not only faster but also comes with low computational cost and high data reduction rate in comparison to an incremental decimation algorithm [19]. For more implementation details of vertex clustering, see the works by Botsch et al. [7, p.113]. Despite being a generally faster algorithm, vertex clustering makes it difficult to control simplified meshes and is more likely to produce non-manifold geometry [7, p.112]. Incremental decimation is a multi-purpose mesh simplification algorithm that can either be set to preserve or modify the topology of a mesh, while vertex clustering is strictly a topology-modifying algorithm [7, p.113]. Non-manifold geometry can simply not be allowed for the runtime rendering and this detail makes the incremental decimation algorithm a more suitable candidate for the proposed approach.

3.1.2 Stage 2

In the pre-computing stage, as described in Algorithm 1, a two-dimensional grid is created containing an open scene with the trianglular meshes. Each target is set to its highest LOD complexity. Each corner of the two-dimensional grid stores four fixed cameras in the cardinal directions where their frustums do not intersect (a field of view of 90 degrees). The process begins by iterating through every camera for each defined corner. In terms of height, the cameras are positioned at the normal height on the Y-axis for a humanoid avatar in a first-person game. An image is rendered and stored along with the LOD combination used.

```
for each corner in the defined grid do
    for for each camera in a corner do
        capture reference image;
        while triangles within view frustum > triangle budget do
            for for each mesh in the view frustum do
                lower LOD version;
                capture image;
                compare image to reference image with SSIM;
                store SSIM difference and difference in vertex count;
                raise LOD version;
            end
            lower mesh with least impact on SSIM;
        end
        store LOD combinations in view frustum;
    end
end
```

Algorithm 1: Pseudocode for the pre-computing stage

Individually each triangular mesh is lowered in LOD version and an image of the

scene is taken. This image is then compared to the reference image using SSIM. The SSIM difference in image quality is stored as well as the difference in vertex count between the switched LOD versions. After each target has been visited, the target that has the least impact on SSIM is lowered in LOD version. The process is repeated for the current frustum until the combined complexity is within half of the budget. In the real-time stage, the two frustum will always be combined to meet the total budget. The process stops when the triangle budget has been met. The LOD versions are then stored in the camera's frustum.

3.1.3 Stage 3

In the real-time stage, the LOD combinations stored in the previous stage will be retrieved. When a camera traverse through the scene it will find the node in the grid closest to the camera, then the two cameras in the closest directions.

3.2 Experiment Design

$$\frac{Cameras \times LODs^{Meshes}}{FPS \times 60^2} \tag{3.1}$$

The tests will use roughly 400 cameras and 40 meshes with five levels of detail for each mesh. Assuming that an average of five meshes is visible at all time from the main camera, this would require approximately 1.250.000 images. Following Equation 3.1, the approximate pre-processing time translated from milliseconds would be around 20 hours or more. It takes into consideration that it is possible to render the scene at 60 frames per second (FPS) and that the average time to compute the SSIM between two images is roughly 76 milliseconds.

It should be noted that this equation returns greater computation times than the current pre-computing stage since the equation takes every possible LOD combination into consideration. The current pre-computing stage only settles with the first LOD combination that qualifies for the triangle budget as illustrated in stage 2. Therefore, the returning value of the equation can be interpreted as the upper bound of all possible computation times. In early tests, it was measured that in two hours we can render and compute SSIM for about 430.000 images, which should be enough for the tests.

The tests use five different triangular models. The models can be seen in Figure 3.1.

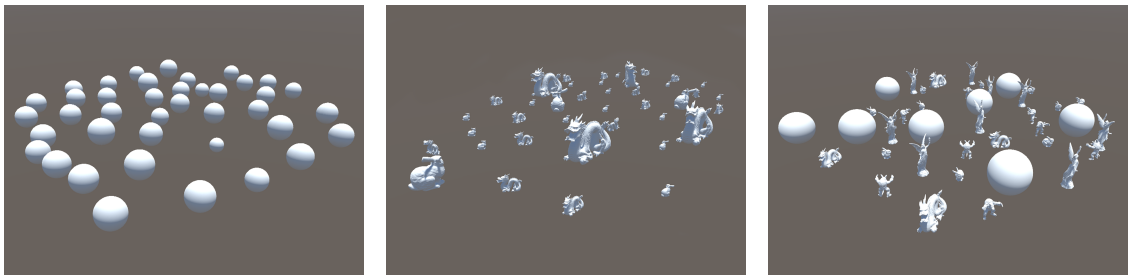
3.2.1 Scenes

Three different open scenes will be used for the experimentation. The motivation behind this is to observe how the image quality-driven LOD selection approach will handle different scenarios. The first scene will have a high object density. The second

(a) (b) (c) (d) (e)

Figure 3.1: Models used for the experiment. All of the models except the "Sphere" come from the Stanford 3D Scanning Repository, a) Armadillo, b) Stanford bunny, c) Dragon, d) Lucy, and e) Sphere.

will have a low object density. The third will have a mixed density, certain parts of the scene are crowded while other parts are scarce. The scenes can be seen in Figure 3.2.



(a) (b) (c)

Figure 3.2: The scenes used for the tests, a) Scene 1, only Sphere, b) Scene 2, Dragon and Stanford Bunny, and c) Scene 3, Sphere, Dragon, Bunny, Armadillo, and Lucy.

3.3 Data Gathering and Measurements

The data gathered from the pre-processing stage will be how much time it will take and how many cameras will be used.

To gather data, an automated test is defined where the test consists of a number of points generated with equal distance between them. These are positioned on the same height as the cameras in the two-dimensional grid along a *bezier curve* forming the camera path. The generated points are stored as permanent reference points and the bezier curve is discarded. The rendering camera following the path will always look in the forward direction of the path. Only rotation around the yaw or the y-axis in Unity will be allowed for the tests.

After the tests have been completed, each image of Unity's existing DLOD approach and the image quality-driven LOD selection approach will be compared to the reference images. Both SSIM and the simplified PDM will be used for image comparison.

The metrics are extracted using an automated script driving the image comparison which presents the results in charts by plotting the data with the command-line driven graphing utility gnuplot [2].

A CPU based implementation of SSIM for the image quality-driven LOD selection approach proved to be insufficient for this research. A single-threaded execution of the conventional SSIM with a forced image resolution of 256x256 pixels demanded a computation time between 5 - 10 seconds. Every second matter when comparing the images in runtime and therefore parallel computations should be considered. This is achieved by transferring all the calculations to the graphical processing unit (GPU). While there are many already existing implementations of SSIM available to use, only a handful of them is running the calculations solely on the GPU.

The Open Source Computer Vision library, more commonly known as OpenCV, includes a GPU module with accelerated code for various image comparison metrics. OpenCV is a popular choice in computer science with its strong focus on real-time applications and its usage ranges from advanced robotics to interactive art. The OpenCV GPU module is written using the parallel computing platform CUDA developed by NVIDIA. CUDA offers a programming model for general computing on the GPU and is easy to incorporate in Unity when using the OpenCV library. SSIM was added to the image quality-driven LOD selection approach by compiling the accelerated code from OpenCV into a dynamic library plugin and later importing it to Unity by making a small wrapper for C#. This became the final choice of SSIM to use for the proposed approach after the CPU implementation.

Computation times for SSIM on the GPU were approximately 65 times faster than the CPU implementation and helped to decrease the pre-processing time. However, it was not verified if the CPU and GPU implementations of SSIM returned similar values when comparing two images. Additionally, they were not tested separately with a third SSIM to fully guarantee that both implementations properly worked which pose a threat of validation to the proposed approach. The GPU implementation of SSIM was assumed to be functional since the expected SSIM value based on visual approximation strongly corresponded with the returning value.

Perceptual Image Difference Utility Settings	
Field of view	85 degrees
Pixel threshold	100 pixels
Gamma	2.2
Luminance	100 candela per meter squared

Table 3.1: Settings used for the perceptual image comparison utility.

The perceptual image comparison utility used to evaluate the image quality produced by the image-quality driven approach is based on the software of previous work in perceptual metrics used at PDI/Dreamworks [33]. When image comparison was performed on rendered movie sequences during production, it was discovered that pixel

changes that were visually insignificant could still result in false positives. This image comparison tool was developed to sort out these false positives from actual defects. The default settings used for this experimentation are seen in table 3.1. By using a field of view of 85 degrees and a pixel threshold of a 100 pixels, the perceptual image comparison utility puts a strict quality constraint on the rendered images where even the slightest noticeable error is detected.

To study the effects of this quality constraint, each rendered image from the image quality-driven LOD selection approach evaluated by the perceptual-metric tests is also presented as a spatiotemporal error tolerance map, which Yee refers to as an Aleph Map. It is constructed from spatiotemporal contrast sensitivity, belonging to both space and time, and presented as a low-level saliency map [34]. The purpose of a saliency map is to simplify and change the representation of an image to make it easier to analyze specific factors of an image [18]. In this case, the saliency map is highlighting the differences in color or luminance from the original reference image to identify where SSIM actually detects structural differences.

Considering the relatively small scale of the image quality-driven approach in comparison to previous work, the simplified PDM was considered to be sufficient for the purpose of this study. It was the first, and only considered choice of PDM at the time which similarly to SSIM pose a threat to validation since it was not tested together with other similar PDM implementations. All that was required to identify the structural changes detected by SSIM was how many pixels changed and where these were located in the image. Therefore, the simplified PDM finds these changes only by measuring the spatial frequency sensitivity, luminance sensitivity, and color sensitivity in an image. Despite the low amount of metrics, these will still provide the necessary details for averaging the perceptually different pixels that are required to analyze the perceivable difference in each scene.

Originally, the resulting metrics from image comparison were manually plotted in gnuplot. However, each executable had to be called separately and this resulted in a more time-consuming extraction of the metrics. With the CUDA based SSIM functions and the PDM software built as two separate executable programs, the extraction was changed to use an automated script to gather the stored images from Unity and call the executables for image comparison when needed. The script starts by taking each frame produced by the image quality-driven LOD selection approach and the built-in approach in Unity to compare them against the corresponding reference image. The resulting image quality metrics are appended to separate lists and sent to gnuplot once all images have been compared. All the commands necessary to plot the data are entered from the script and the resulting charts are exported as vector graphics. Since the resulting image metrics are no longer manually plotted from files stored on the disk, temporary files are created in memory from the lists using named data blocks.

3.4 Experimental Setup

The apparatus used for both the pre-computational stage and the runtime tests are seen in Table 3.2. The apparatus is on the high-end spectrum of personal computers, and it should not pose a hindrance to the general performance of the application.

The setup and pre-processing time for each scene can be seen in Table 4.2. Each scene has a total of 200 meshes including all of the LOD versions. The triangle budget used for the scenes was 300000.

Desktop setup	
OS	Windows 10 Education 64-bit (10.0 Build 15063)
RAM	16384MB
CPU	Intel(R) Xeon(R) CPU ES v4 @ 3.50GHz
GPU	NVIDIA GeForce GTX 1080

Table 3.2: The computer hardware specifications used for the tests.

Chapter 4

Results

The tests were executed with 1000 images being captured in each LOD approach with a resolution of 1920x1080 pixels. The thresholds used for Unity's built-in approach was: *0.5, 0.25, 0.125, 0.075,* and *0.01*. These threshold values are the screen relative heights to use for the transition in the range 0 to 1. By interpreting the first LOD threshold value, the first LOD transition will occur when an object covers less than 50% of the relative screen height.

4.1 Pre-Processing

Statistics for the pre-processing includes the number of triangles generated by the incremental decimation algorithm for each LOD version and the total pre-processing time for each scene. The number of vertices for each LOD version is seen in Table 4.1 and the time for the pre-processing stage is seen in table 4.2.

Vertices					
	LOD 0	LOD 1	LOD 2	LOD 3	LOD 4
Armadillo	64860	32430	16214	8106	4052
Bunny	69666	34832	17416	8708	4354
Dragon	55000	27500	13750	6874	3436
Lucy	59998	29998	17120	14452	13898
Sphere	28560	14380	7140	3570	1784

Table 4.1: Number of triangles for each LOD version of the test models. LOD 0 is the original mesh.

Scenes			
	Time	Unique Meshes	Cameras
Scene 1	13h 25min	5	404
Scene 2	30h 11min	10	378
Scene 3	15h 39min	25	284

Table 4.2: Pre-processing time for each scene. Cameras which do not include any geometry are culled.

4.2 Runtime Tests

A series of runtime tests were performed to demonstrate the framework of the proposed approach in three different scenes. In the following sections, the statistics for both SSIM quality and perceptual pixel difference in the scenes are summarized. The results for the perceptual pixel difference are seen in Table 4.3, as well the resulting SSIM quality in Table 4.4.

Pixel difference						
	Unity			Framework		
	Max	Min	Avg	Max	Min	Avg.
Scene 1	1150	100	354	5854	100	346
Scene 2	347407	259	11483	472983	195	29134
Scene 3	418481	187	31668	500579	180	32052

Table 4.3: Pixel difference in each scene compared to the reference.

SSIM quality						
	Unity			Framework		
	Max	Min	Avg	Max	Min	Avg
Scene 1	1	0.998774	0.999579	1	0.999172	0.999916
Scene 2	1	0.971588	0.995292	1	0.950642	0.989877
Scene 3	1	0.932154	0.988967	1	0.919427	0.988706

Table 4.4: SSIM quality in each scene compared to the reference.

4.2.1 Runtime Test Scenes Data

Values are averaged within the scope of the standard deviation to give a fair approximation of the data.

The first scene only included instances of the Sphere test model. The SSIM results are seen in Figure 4.1, Figure 4.2, and Figure 4.3. The PDM results are seen in Figure 4.4, 4.5, and 4.6.

The second scene only included instances of the Bunny and Dragon test models. The SSIM results are seen in Figure 4.7, Figure 4.8, and Figure 4.9. The PDM results are seen in Figure 4.10, Figure 4.11, and 4.12.

The third test scene included instances of all test models. The SSIM results are seen in Figure 4.13, Figure 4.14, and Figure 4.15. The PDM results are seen in Figure 4.16, Figure, 4.17, and Figure 4.18.

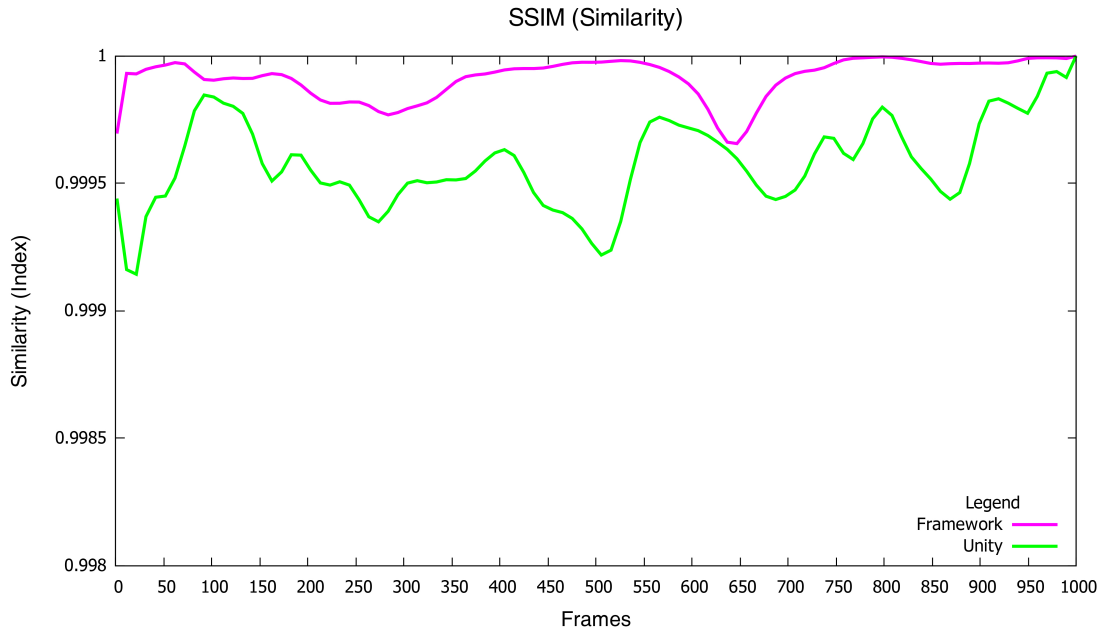


Figure 4.1: Averaged SSIM index comparison in scene 1, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows SSIM value for both solutions in each frame.

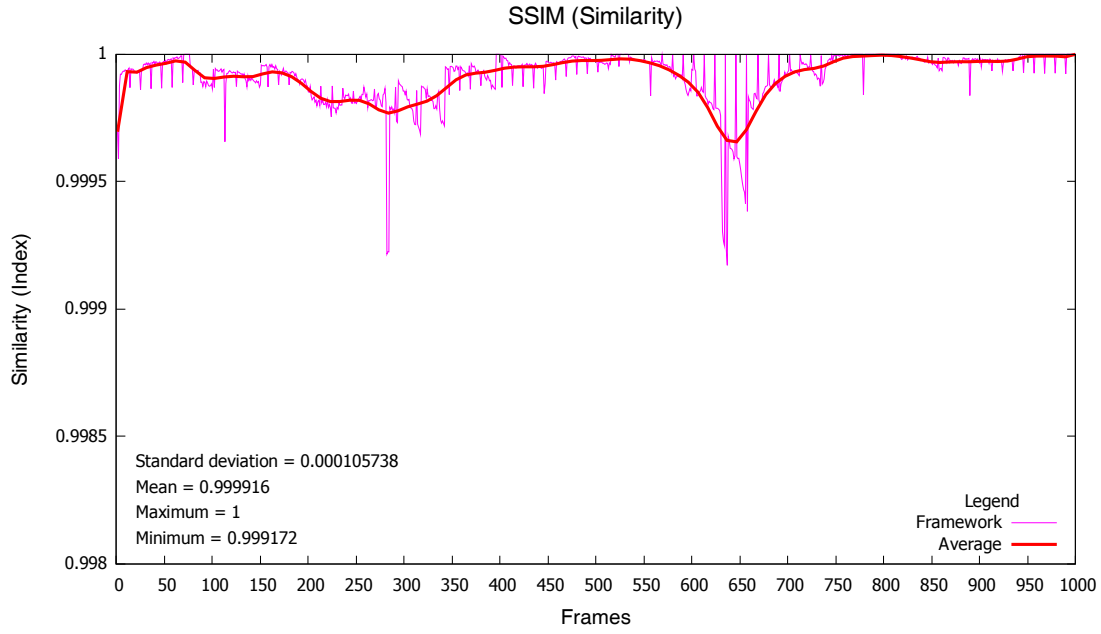


Figure 4.2: SSIM index for the proposed approach in scene 1, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows SSIM value in each frame.

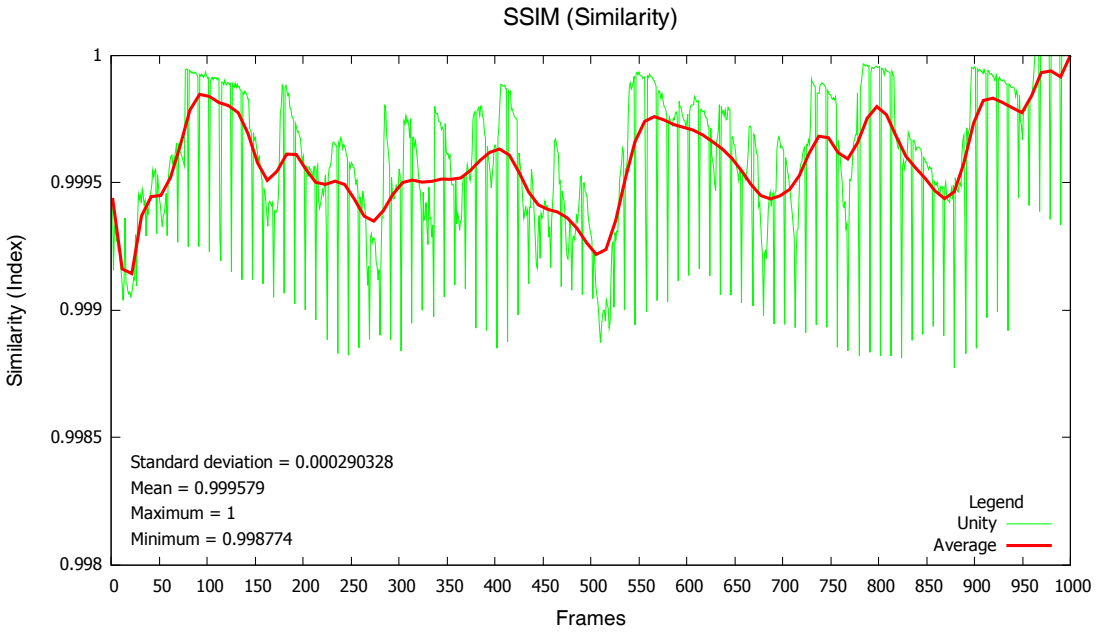


Figure 4.3: SSIM index for Unity in scene 1, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows SSIM value in each frame.

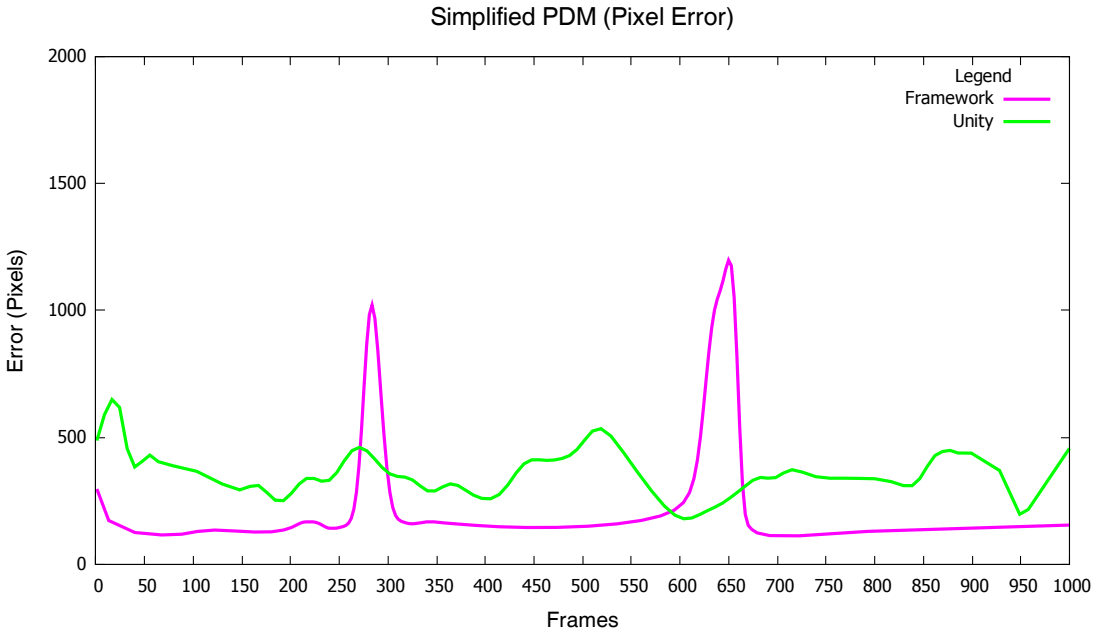


Figure 4.4: Averaged perceptual pixel difference comparison for scene 1, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows perceptual pixel difference for both solutions in each frame.

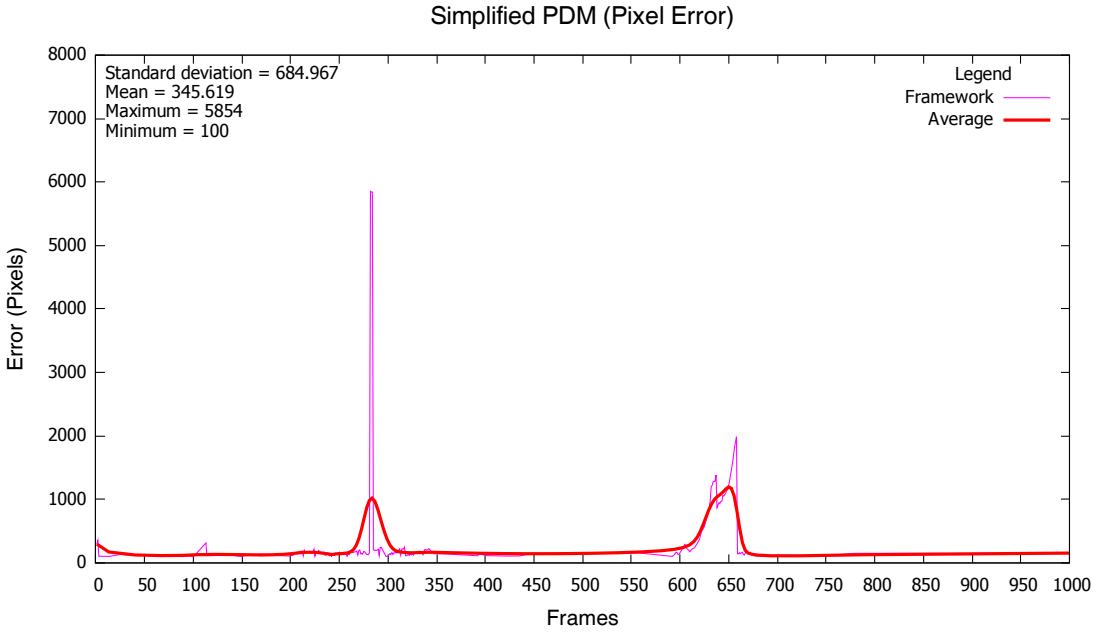


Figure 4.5: Perceptual pixel difference for the proposed approach in scene 1, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows perceptual pixel difference in each frame.

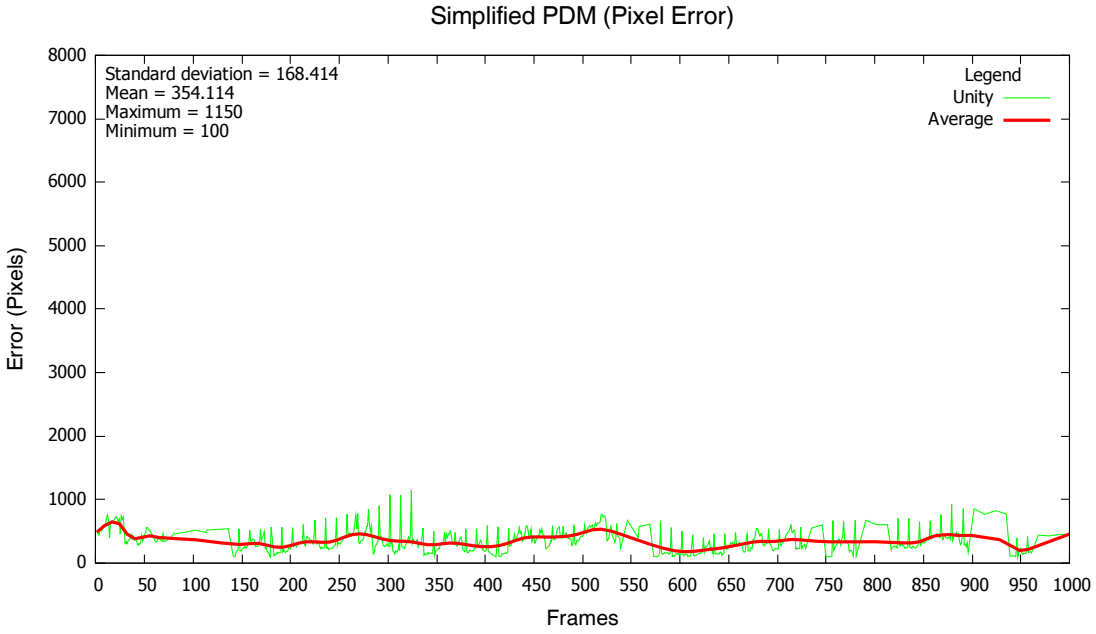


Figure 4.6: Pixel difference for Unity in scene 1, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows perceptual pixel difference in each frame.

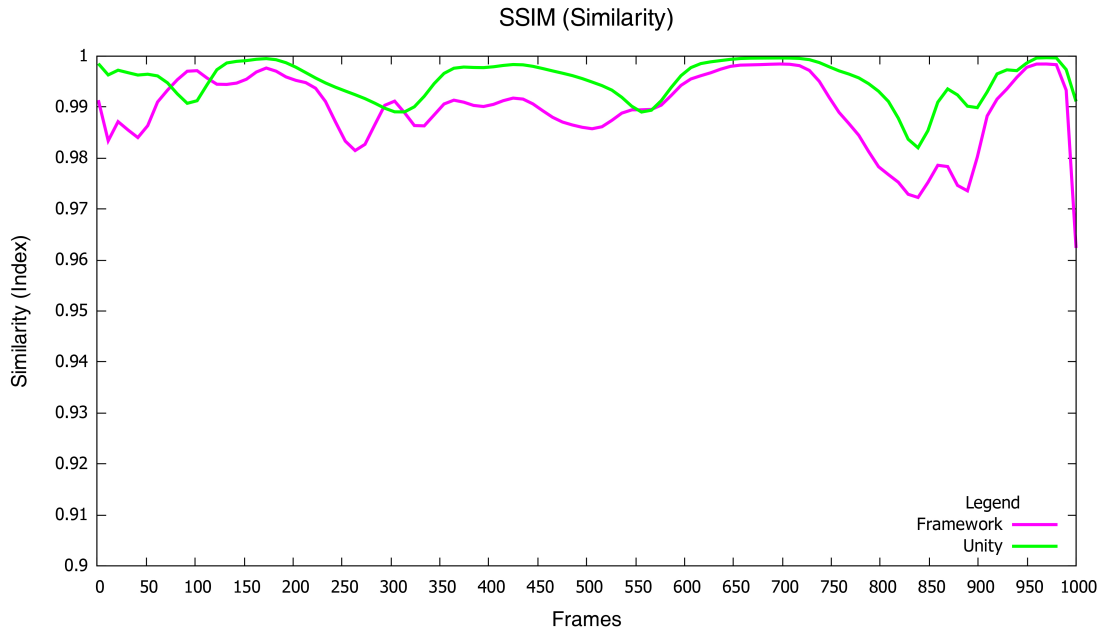


Figure 4.7: Averaged SSIM index comparison for scene 2, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows SSIM for both solutions in each frame.

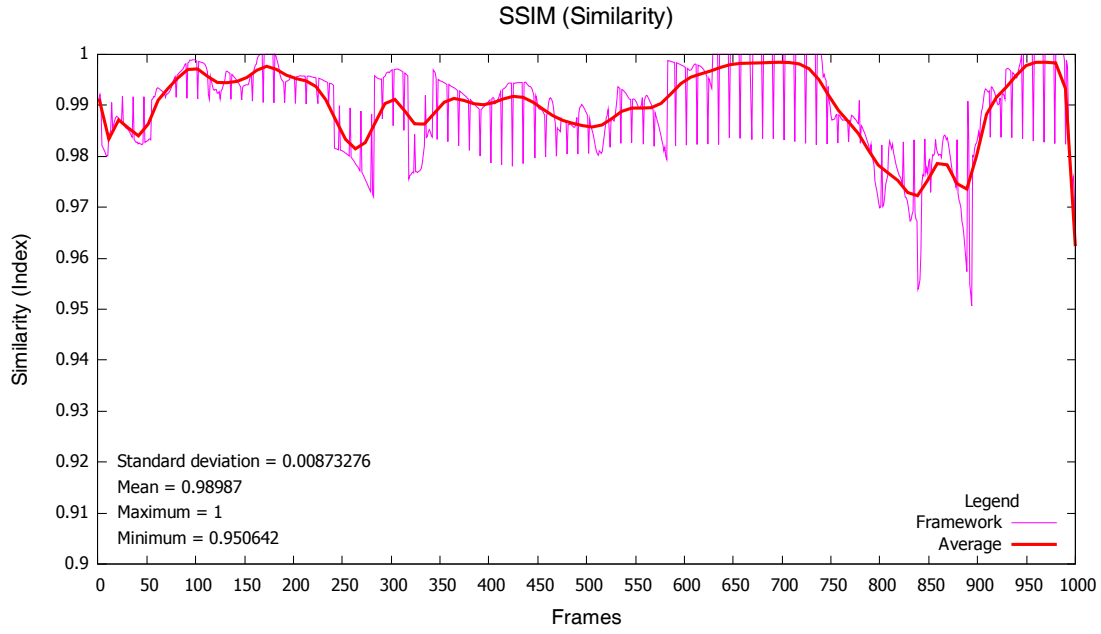


Figure 4.8: SSIM index for the proposed approach in scene 2, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows SSIM in each frame.

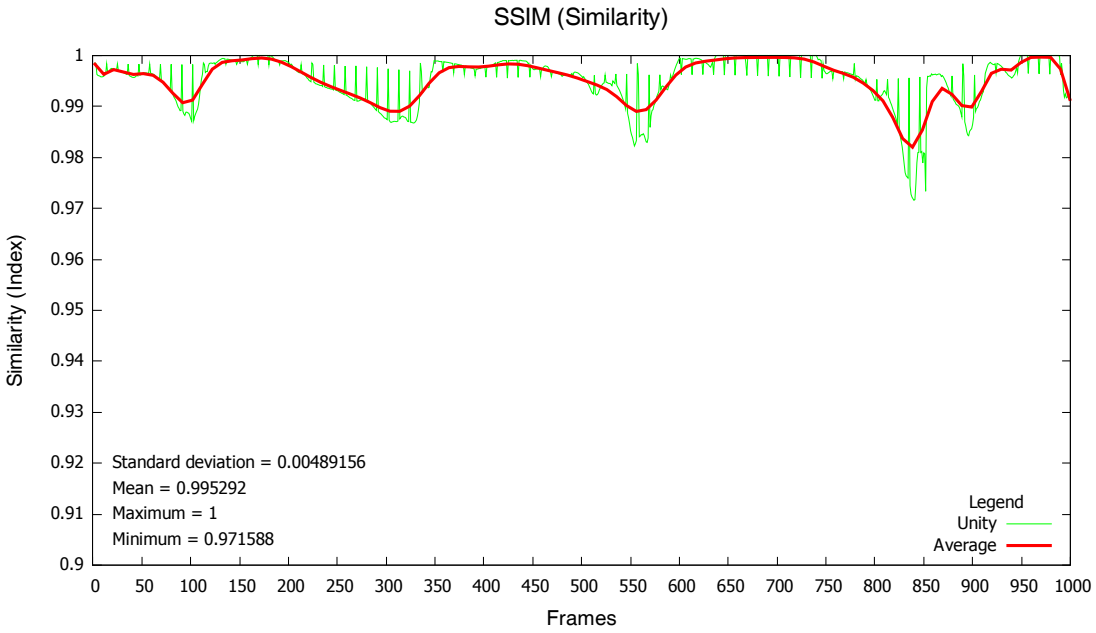


Figure 4.9: SSIM index comparison for Unity in scene 2, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows SSIM in each frame.

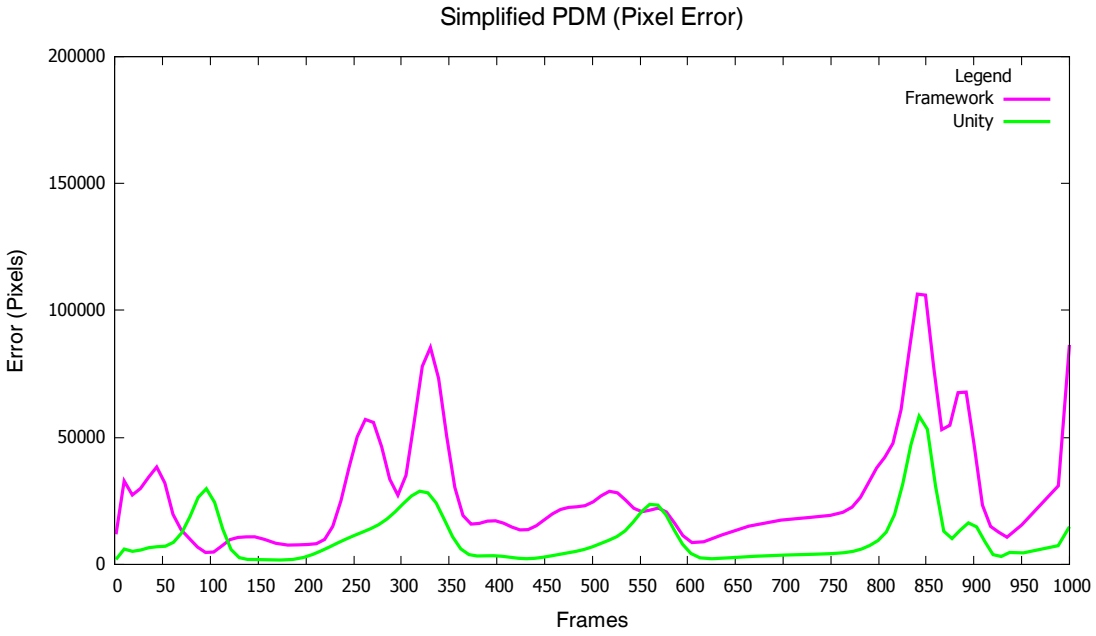


Figure 4.10: Averaged perceptual pixel difference comparison for scene 2, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows perceptual pixel difference for both solutions in each frame.

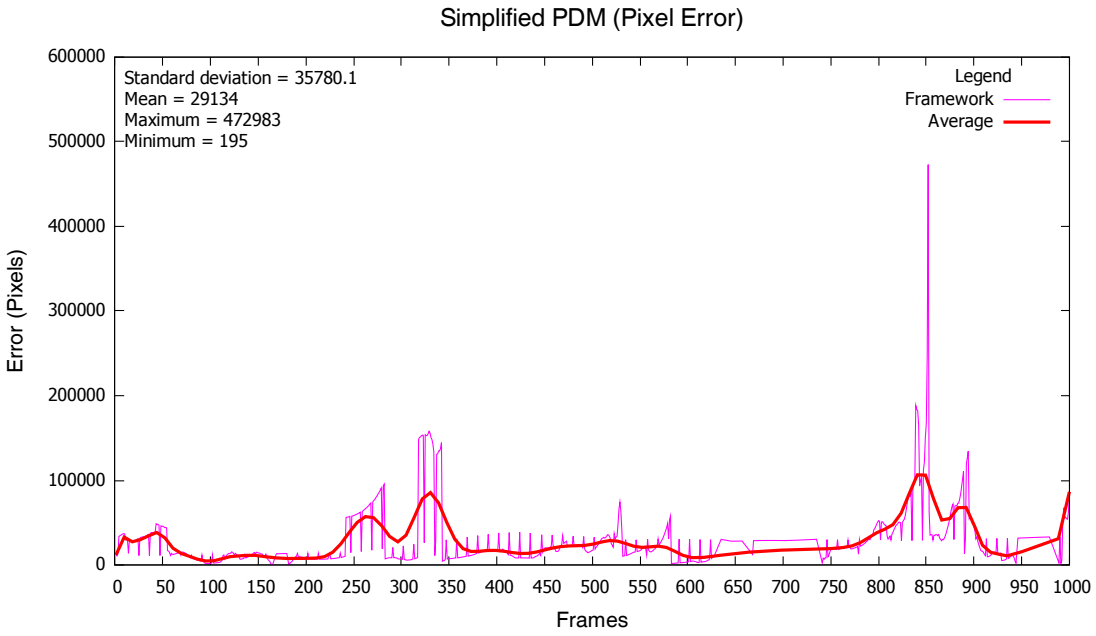


Figure 4.11: Perceptual pixel difference in framework for scene 2, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows perceptual pixel difference in each frame.

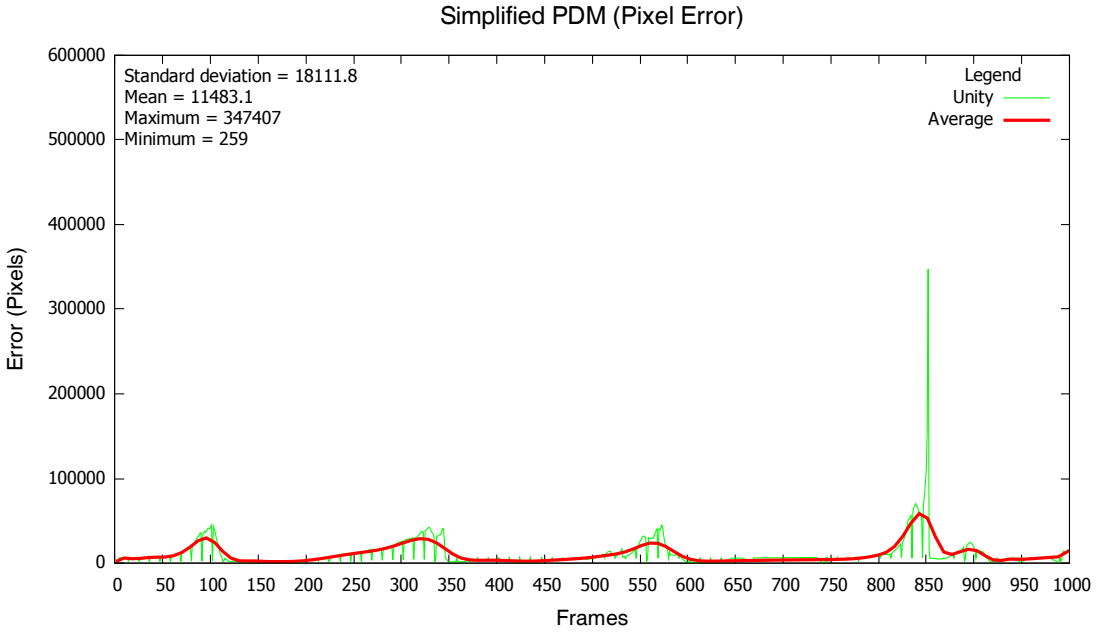


Figure 4.12: Perceptual pixel difference in Unity for scene 2, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows perceptual pixel difference in each frame.

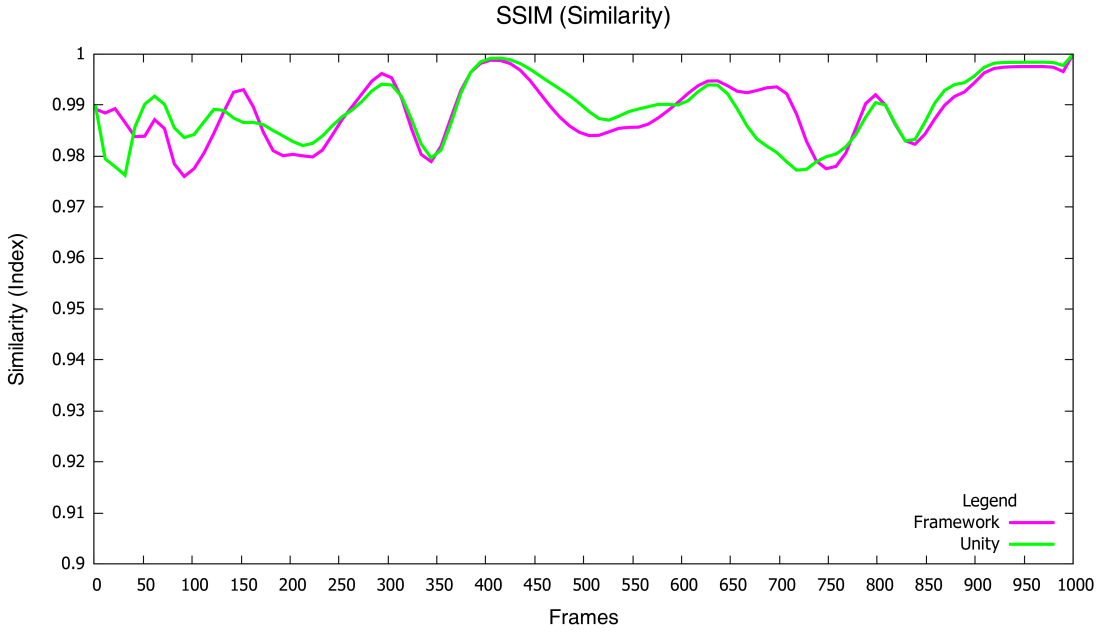


Figure 4.13: Averaged SSIM index for the proposed approach and Unity in scene 3, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis SSIM for both solutions in each frame.

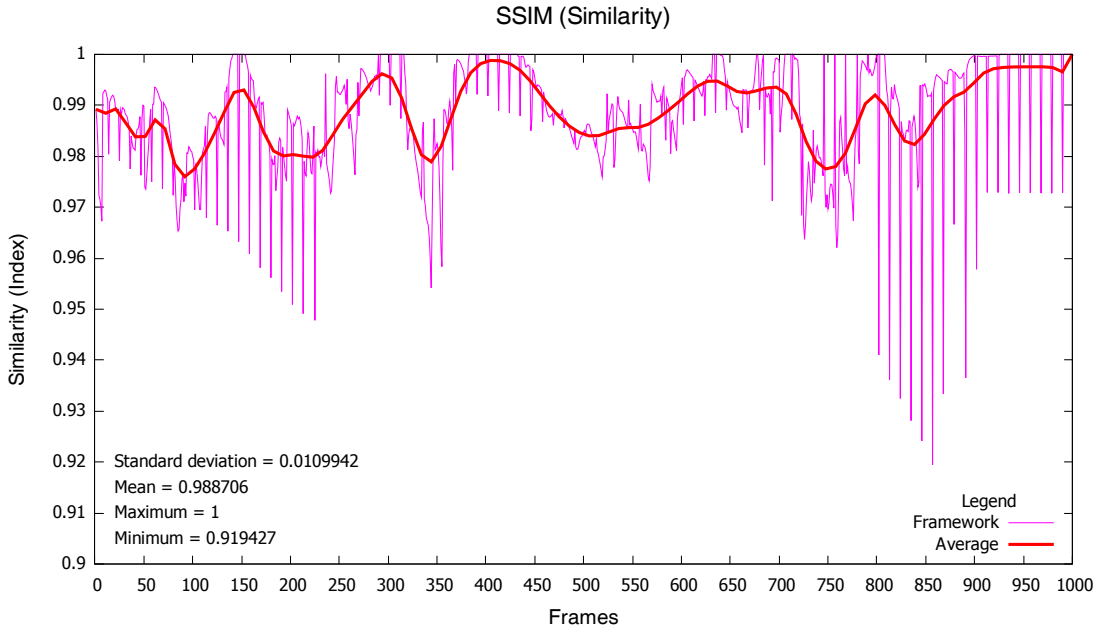


Figure 4.14: SSIM index for the proposed approach in scene 3, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows SSIM in each frame.

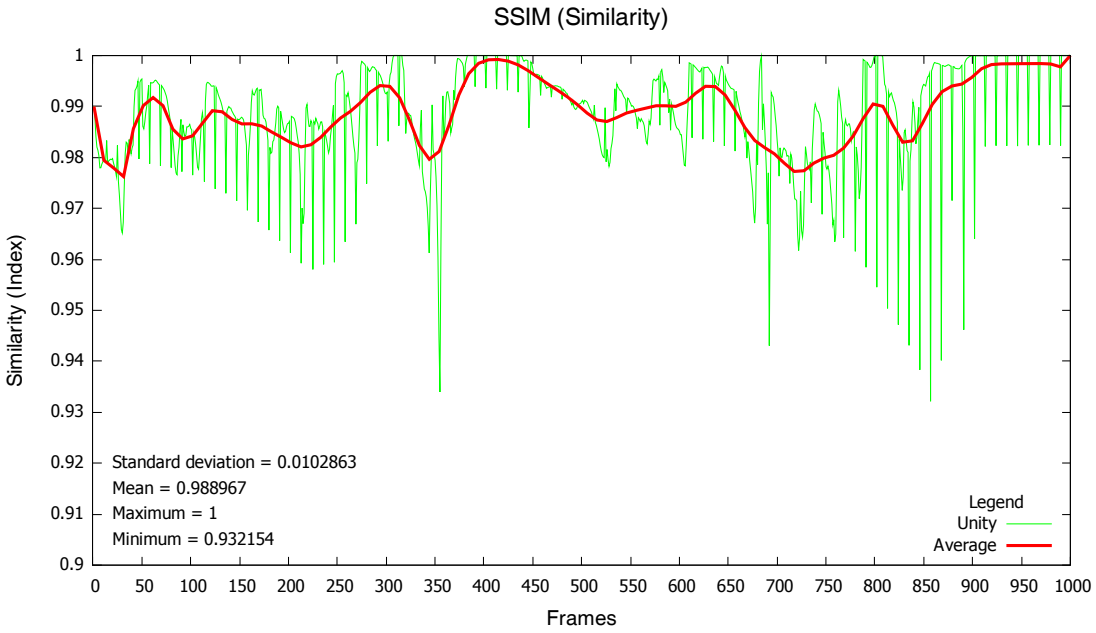


Figure 4.15: SSIM index for Unity in scene 3, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows SSIM in each frame.

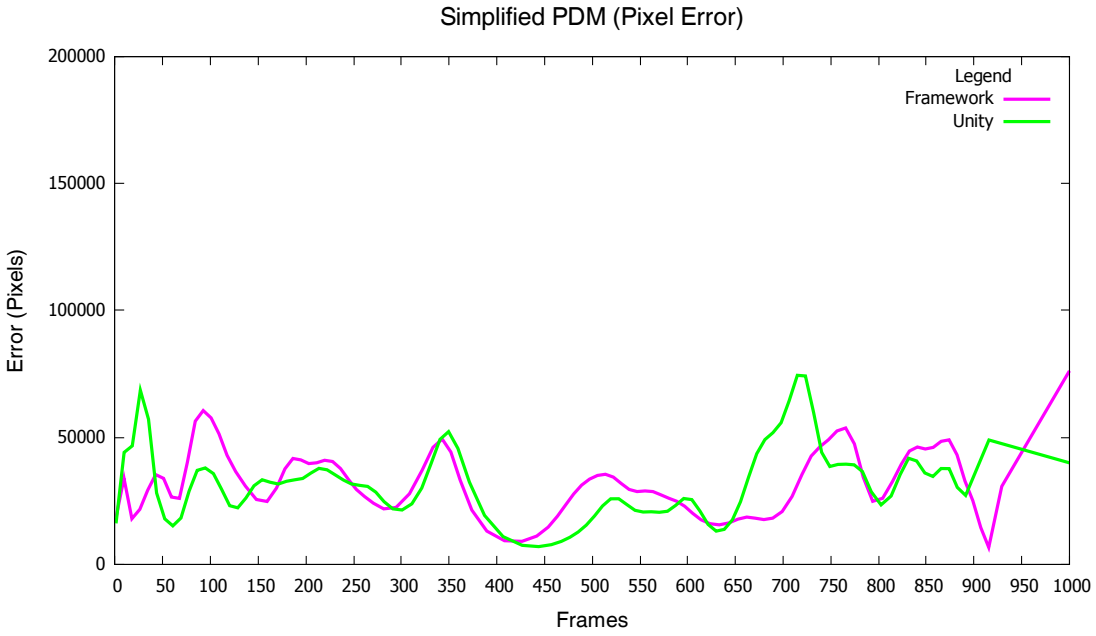


Figure 4.16: Averaged perceptual pixel difference for scene 3, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows perceptual pixel difference for both solutions in each frame.

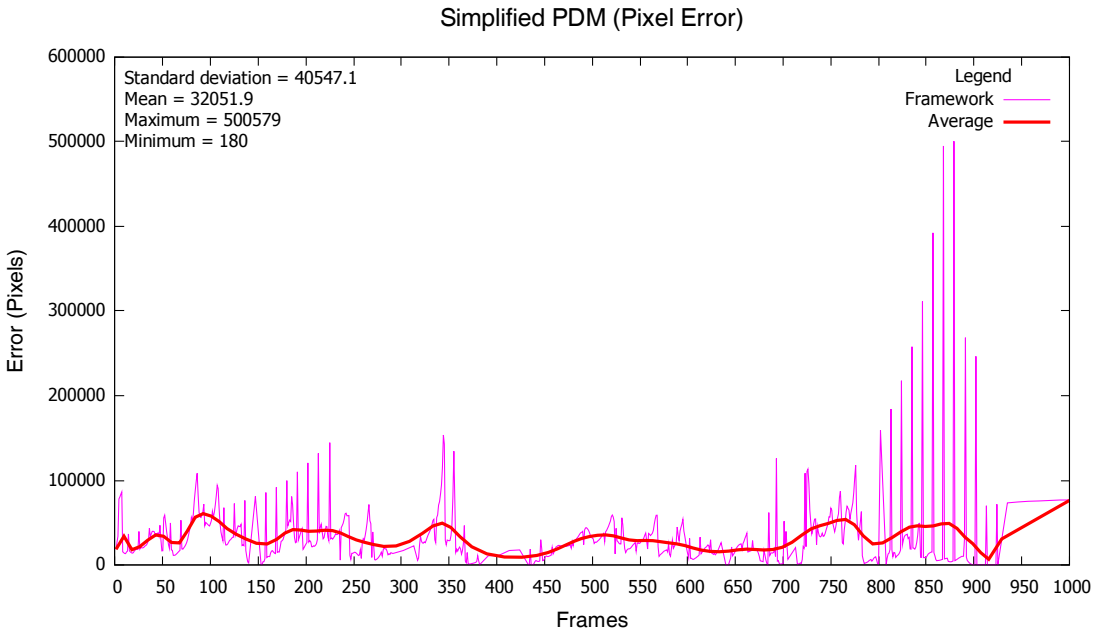


Figure 4.17: Perceptual pixel difference for the proposed approach in scene 3, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows perceptual pixel difference in each frame.

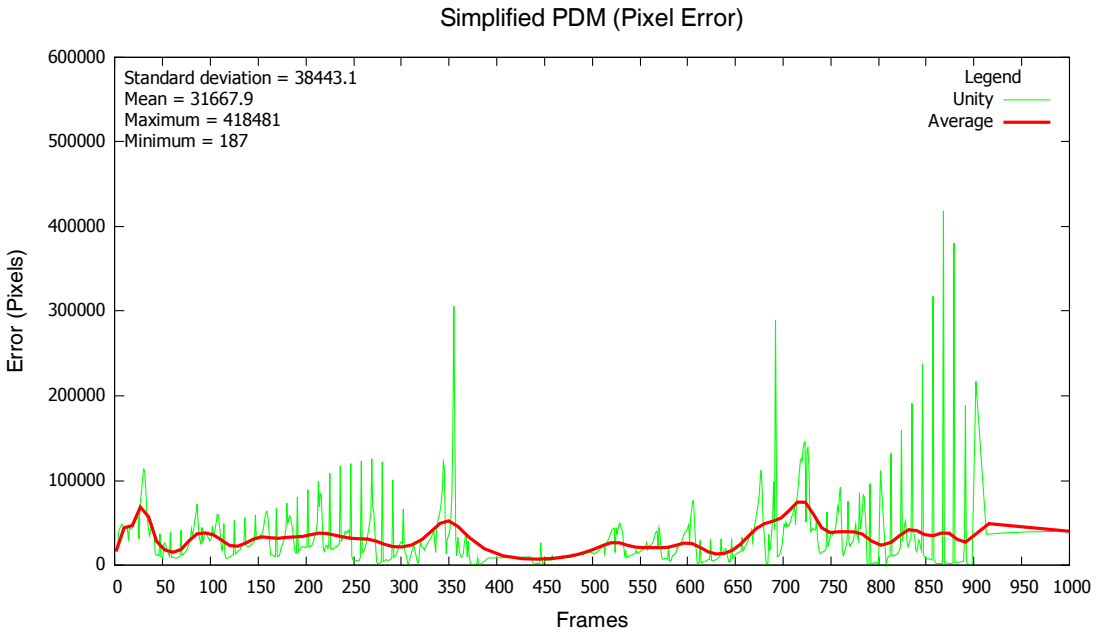


Figure 4.18: Perceptual pixel difference for Unity in scene 3, for a 1000 frames camera movement. The X-axis illustrates the number of captured frames, while the Y-axis shows perceptual pixel difference in each frame.

Chapter 5

Analysis and Discussion

The focus of this analysis is to investigate the rare cases where large differences between the images were detected and how the visible outliers in the results affect the output of the data. Despite that Unity produced better SSIM quality and average SSIM in almost every case, the image quality-driven LOD selection approach was capable of producing a similar SSIM quality. The largest difference between the highest recorded SSIM quality for both solutions was found in scene 2. The image quality-driven LOD selection approach produced a slightly better SSIM quality in scene 1, where it merely surpassed Unity in SSIM quality.

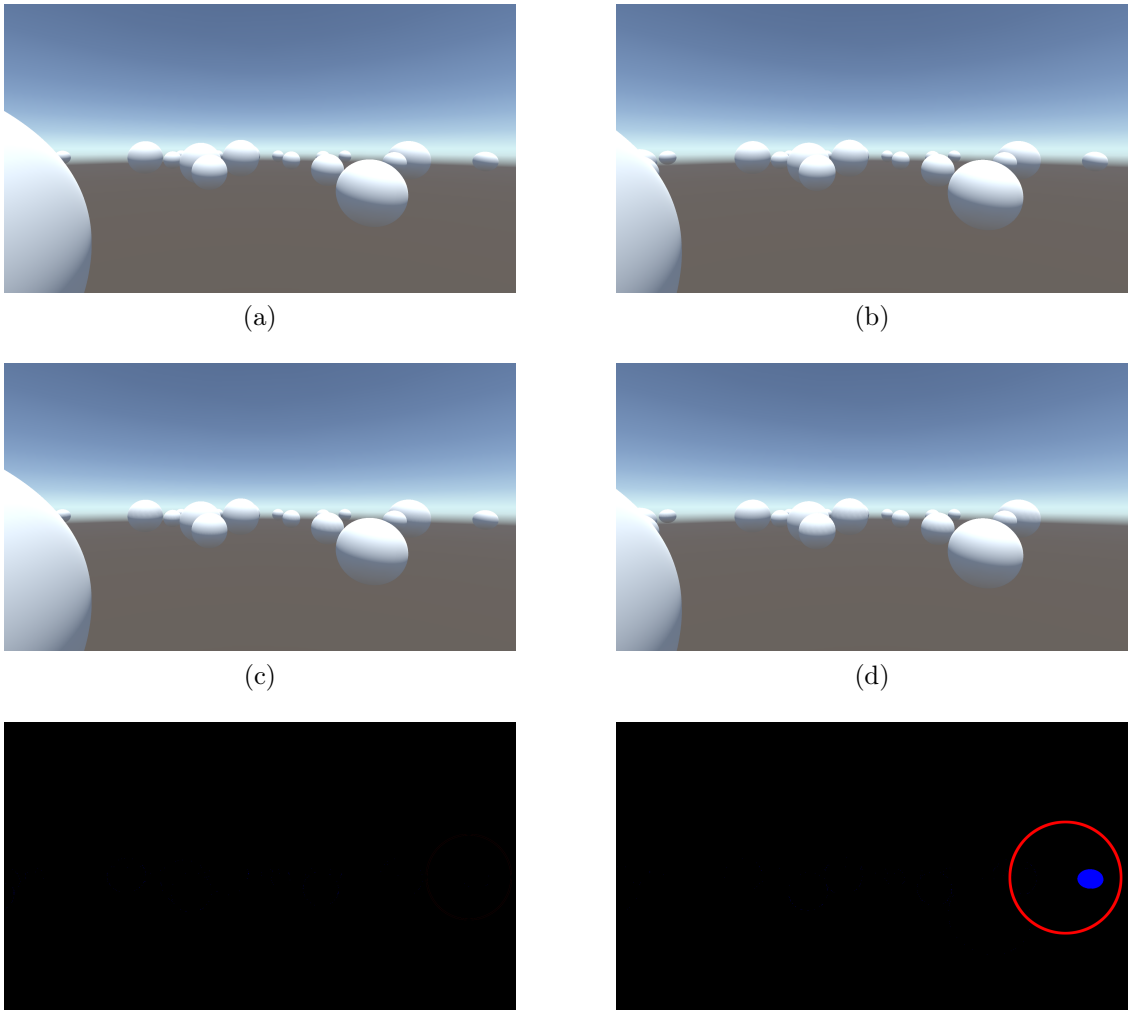
Unity produced less pixel difference for every scene including an overall lower average except for scene 1. By also analyzing the pixel difference for each scene, there appears to be a clear connection between the metrics in both solutions. This relationship was explored using the Pearson correlation coefficient to measure the strength of the linear association between the two variables [3]. The closer this association is to 1 or -1, the stronger the relationship is between the variables. In this case, an increase in SSIM quality decreases the pixel difference, hence the value should be closer to a downhill linear relationship of -1. Furthermore, the magnitude of the relationship is received by calculating the square root of the coefficient. The resulting Pearson correlation coefficient and its magnitude for each scene are seen in Table 5.1.

Data	Pearson Correlation	Magnitude
Scene 1	- 0.746	86 %
Scene 2	- 0.761	87 %
Scene 3	- 0.854	92 %

Table 5.1: Pearson correlation coefficient depicting the relationship between SSIM quality and pixel difference for each scene.

How strongly the SSIM quality is affected by the pixel difference depends on how structurally similar the images remain after a change is introduced. Large amounts of pixel difference caused by the loss of visual details from a coarser geometry can still return a high SSIM quality if the original shape of the object is preserved. Since the incremental decimation algorithm successfully preserves the original shape of the mesh silhouettes, the changes in SSIM quality are less noticeable unless a smaller range closer to 1 is investigated.

5.1 Analysis Scene 1



(a) (b) (c) (d) (e) (f)

Figure 5.1: Pixel difference in the framework highlighted by blue for scene 1, a) Reference frame 284, b) Reference frame 285, c) Framework frame 284, d) Framework frame 285, e) Frame 284 difference, and f) Frame 285 difference.

The least amount of pixel difference was observed in scene 1 which only used instances of the Sphere test model. The pixel difference was commonly only detected in the silhouettes of spheres using a lower LOD version. The visual impact caused by the coarser geometry on the original shape of the spheres was nearly imperceivable, but enough to break the pixel threshold of 100 pixels in the perceptual image difference utility. The fluctuation in SSIM quality was more noticeable in Unity compared to the image quality-driven LOD selection approach as seen in Figure 4.3.

One noticeable peak in pixel difference was found in between frames 284 and 285 in Figure 5.1, where one sphere in the right side of the view was accidentally culled and reappeared in the next frame. This resulted in a drop from 5841 to only 210 different pixels, which is a total difference of 5631 pixels causing an unexpected deviation in

the data. This single large peak beyond the standard deviation in pixel difference would normally be described as outliers caused by errors in the data. However, the pixel difference for the frame is valid but was not expected to occur due to a rare issue of accidental culling.

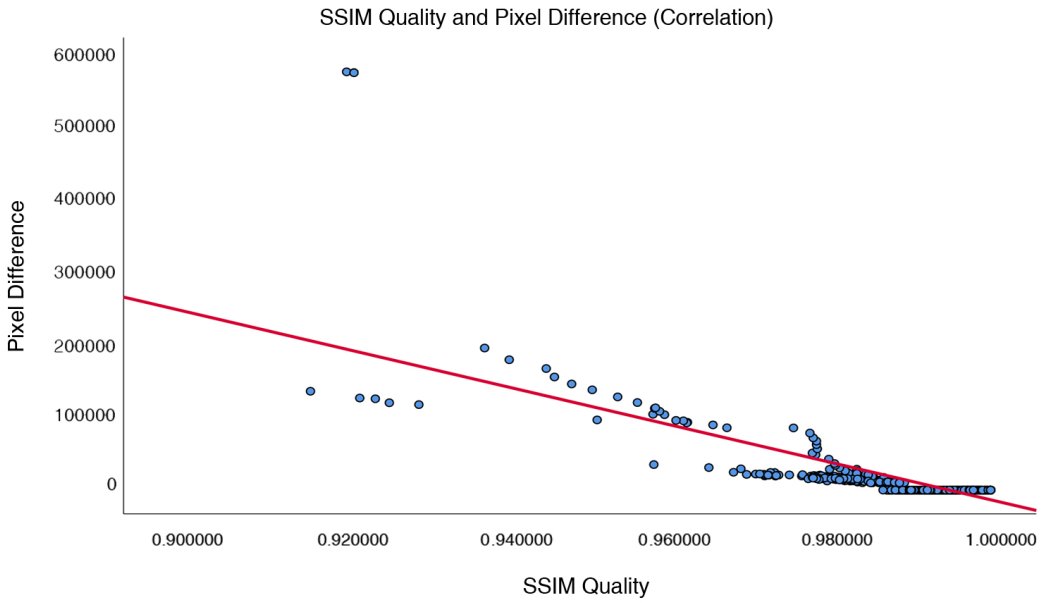
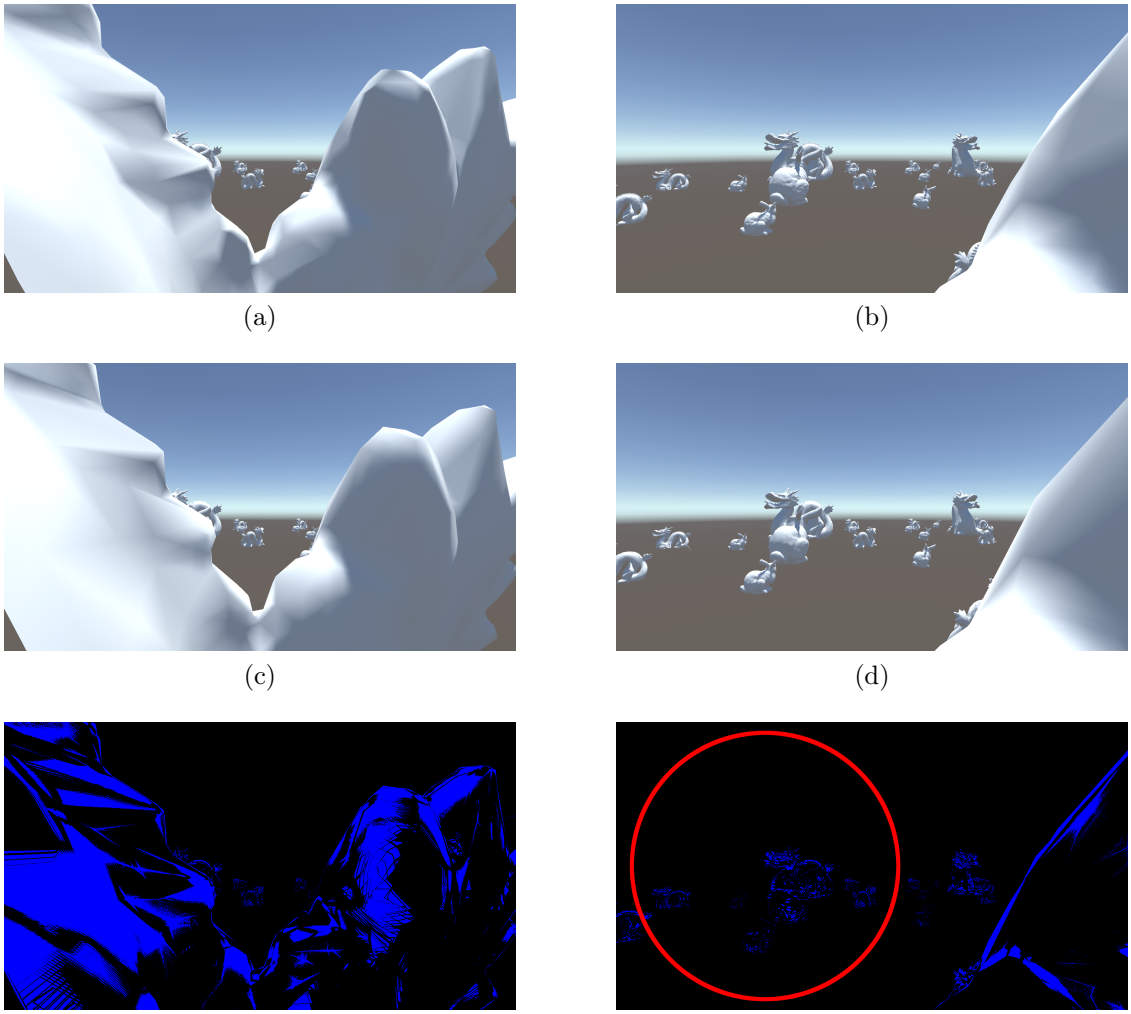


Figure 5.2: Correlation between SSIM quality and perceptual pixel difference for the proposed approach in scene 1.

Correspondingly the SSIM quality was increased when the pixel difference decreased, as seen in Figure 5.2. It never went below 0.999 since the overall shape of the spheres was still structurally similar. 86% of the variance in SSIM quality is explained by the pixel difference, which depicts a strong relationship between the two variables.

5.2 Analysis Scene 2



(a) (b) (c) (d) (e) (f)

Figure 5.3: Pixel difference in the framework highlighted by blue for scene 2, a) Reference frame 852, b) Reference frame 853, c) Framework frame 852, d) Framework frame 853, e) Frame 852 difference, and f) Frame 853 difference

Due to the fact that scene 2 was the most complex scene by only using instances of the two most expensive test models, namely the Dragon and Bunny test models, it features the largest recorded pixel difference in all scenes. One of the largest peaks in pixel difference for both Unity and the image quality-driven LOD selection approach reaching far beyond the standard deviation was found in the interval between frames 850 and 860. Observations made of the captured images from this interval found that the main camera was passing through a dragon test model that did not entirely match the reference.

As the main camera approached the dragon, pixel difference started to increase almost exponentially since the dragon occupied larger regions of the screen. Unity's built-in solution matches the dragon better with the reference image as seen in Figure

4.12 compared to the image quality-driven LOD selection approach seen in Figure 4.11, but suffers from the same issue causing the exponential growth in pixel difference. Once the camera passed the dragon between frames 852 and 853 in Figure 5.3 using the image quality-driven LOD selection approach, the number of different pixels dropped from 472983 to 63037 in only one frame. That is a total difference of 409946 pixels, which explains the large peak in the chart.

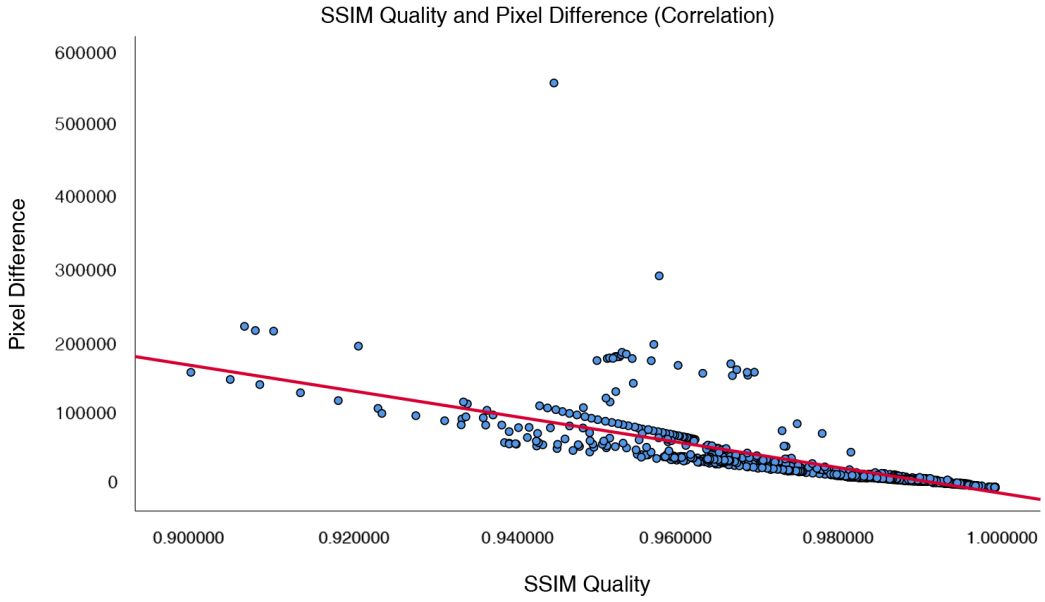
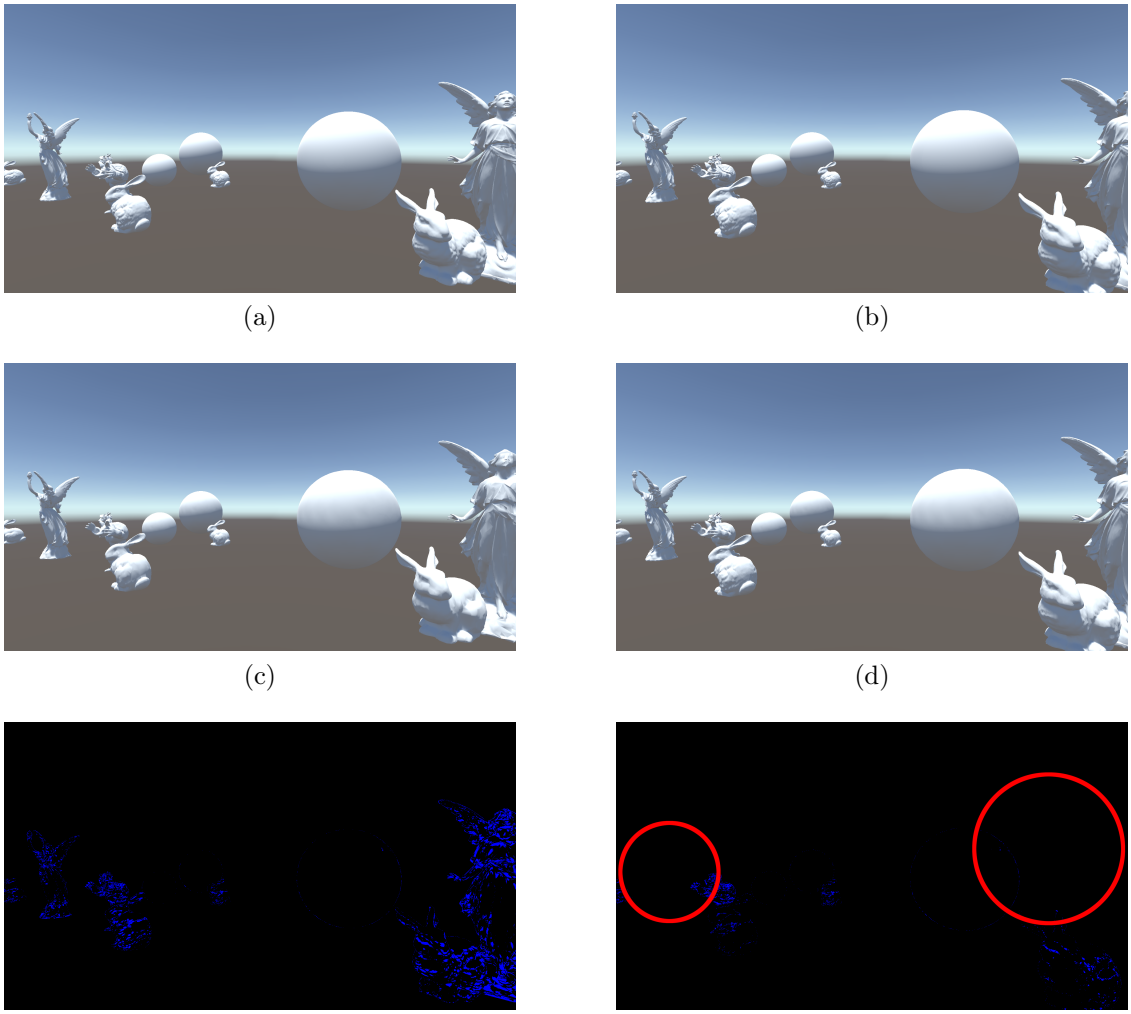


Figure 5.4: Correlation between SSIM quality and perceptual pixel difference for the proposed approach in scene 2.

The SSIM quality and pixel difference for both solutions followed nearly the same pattern throughout the runtime tests. The only difference was that Unity's pixel difference was less aggressive and therefore SSIM quality turned out higher compared to the image quality-driven LOD selection approach. Similarly to scene 1, the SSIM quality increased when the pixel difference decreased as seen in Figure 5.4. 87% of the variance in SSIM quality is explained by the pixel difference, which shows yet another example of a strong relationship between the two variables.

5.3 Analysis Scene 3



(a) (b) (c) (d) (e) (f)

Figure 5.5: Pixel difference in the framework highlighted by blue for scene 3, a) Reference frame 857, b) Reference frame 858, c) Framework frame 857, d) Framework frame 858, e) Frame 857 difference, and d) Frame 858 difference

The most frequent and aggressive recorded peaks in pixel difference for both the image quality-driven LOD selection approach and Unity were discovered in scene 3 for the entire interval between the frames 800 and 900. A closer inspection revealed that models visible in this interval switched more frequently between their LOD versions.

This happened to be three Bunny and two Lucy test models. Note that the Bunny is the most complex model in terms of triangle count, thus making it an expensive frame to render. Because of the complex topology of the Lucy model, the incremental decimation algorithm was unable to lower the number of triangles by half in the last two LOD versions without producing non-manifold geometry. The reduction was halted and only applied where possible to maintain a manifold geometry, resulting in a more complicated mesh in terms of triangles at the lowest LOD version com-

pared to the other models. Since the LOD switching repeats itself for the Lucy and Bunny models in the same interval more frequently to meet the triangle budget, it results in a greater standard deviation in the data. This is because the pixel differences are more noticeable when these two models in particular are switching LOD versions.

Between the frames 857 and 858 in Figure 5.5, the two visible Lucy models matched with the reference image using the image quality-driven LOD selection approach. A few frames later, these disappeared out of the visible frustum and caused a large number of pixel differences to vanish in comparison to the previous frames. The same effect is also visible for Unity's built-in LOD solution in the same interval but appears to be less aggressive which explains the smaller peaks as seen in Figure 4.18. Most of the pixel difference in the models originate from a loss of central visual details introduced by a lowered triangle count, for example in Lucy's clothing and the Bunny's fur.

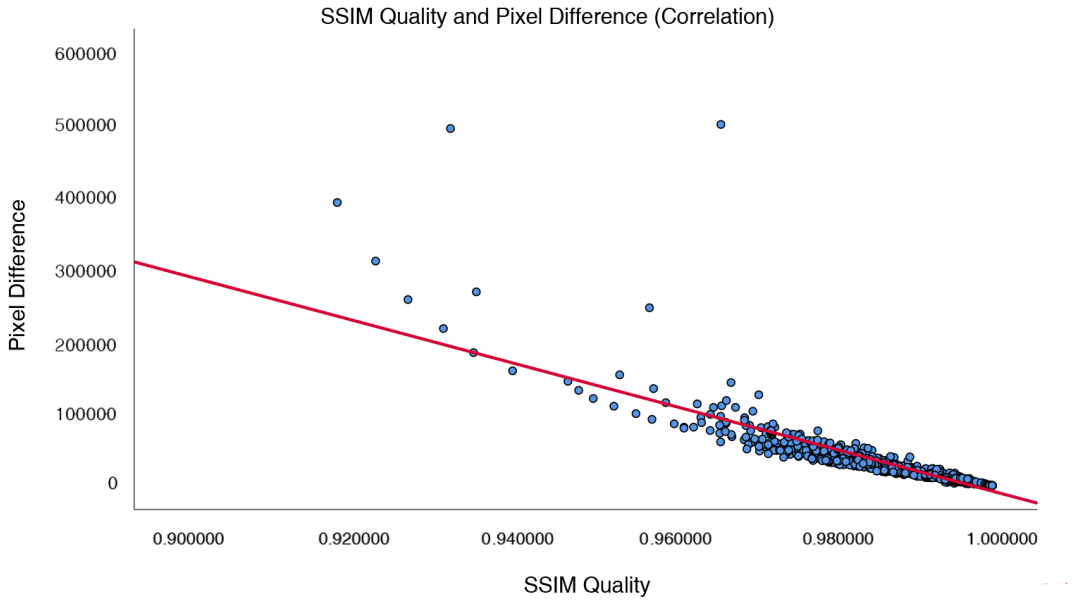


Figure 5.6: Correlation between SSIM quality and perceptual pixel difference for the proposed approach in scene 3.

As a result of the large pixel difference, the SSIM quality produced by both solutions was the lowest recorded for all scenes. The SSIM quality continues to increase in this scene as well when the pixel difference is decreased as seen in Figure 5.6. 92% of the variance in SSIM quality is explained by the pixel difference, which makes it the strongest example of the relationship between the two variables.

Chapter 6
Conclusions and Future Work

The conclusion drawn from the experiment is that when comparing the SSIM quality of rendered images between Unity's built-in approach to LOD and the proposed approach, Unity's built-in approach generally performed better in terms of SSIM.

However different settings, for example, a shorter distance between the corners in the grid could result in better quality. Other factors, such as textures and different lighting could change the results. Given these results, it would not be feasible to use the proposed approach because of the additional cost of precomputing all cameras for a given scene.

6.1 Future Work

In its current state, the image quality-driven LOD selection approach settles with the first LOD combination for a given view that satisfies the triangle budget. For further investigation, the image quality-driven LOD selection approach could be extended to find the best possible SSIM quality for all the possible LOD combinations given a triangle budget using a brute-force search. That way it is guaranteed that no LOD combinations that satisfy the triangle budget, or potentially provide an even better SSIM quality are left out.

After having successfully tested the image quality-driven LOD selection approach using a two-dimensional grid, the next step in the implementation would be to use an octree. It would enable the main camera to traverse the scene on different heights and allow it to rotate around all three axes. However, it would come to the price of an exponential increase in the cost of the pre-computing stage. This process could be made more parallel to reduce the pre-processing times. One computer can be dedicated for rendering while a group of other computers is simultaneously running the SSIM-based selection and slowly adapting its settings to the runtime rendering.

References

[1] Fast-Quadric-Mesh-Simplification. https://github.com/sp4cerat/Fast-Quadric-Mesh-Simplification, 2015. [Online; accessed 9-June-2018].

[2] gnuplot - Official gnuplot documentation. http://www.gnuplot.info/documentation.html, 2018. [Online; accessed 9-June-2018].

[3] SPSS Tutorials: Pearson Correlation. "https://libguides.library.kent.edu/SPSS/PearsonCorr, 2018. [Online; accessed 9-June-2018].

[4] Unity - Manual: LOD Group. https://docs.unity3d.com/Manual/LevelOfDetail.html, 2018. [Online; accessed 9-June-2018].

[5] UnityMeshSimplifier. https://github.com/Whinarn/UnityMeshSimplifier, 2018. [Online; accessed 9-June-2018].

[6] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-time rendering*. AK Peters/CRC Press, Boca Raton, FL, USA, 2008.

[7] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. CRC press, Boca Raton, FL, USA, 2010.

[8] Martin Čadík and Pavel Slavík. Comparing image processing operators by means of the visible differences predictor. In *WSCG (Posters)*, pages 37–40, 2004.

[9] David Cage. Quantic Dream's PS4 Tech Demo. https://www.youtube.com/watch?v=Tw1l_C4kXrg, 2013. [Online; accessed 9-June-2018].

[10] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification envelopes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 119–128, New York, NY, USA, 1996. ACM.

[11] Isabelle Forsman. Automatic LOD selection. Master thesis, Linköpings Universitet, Faculty of Computing and Electrical Engineering, 2017.

[12] Xinbo Gao, Wen Lu, Dacheng Tao, and Xuelong Li. Image quality assessment and human visual system. In *Visual Communications and Image Processing 2010*, volume 7744, page 77440Z, Bellingham, WA, USA, 2010. International Society for Optics and Photonics.

[13] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[14] Markus Giegl and Michael Wimmer. Unpopping: Solving the image-space blend problem for smooth discrete lod transitions. In *Computer Graphics Forum*, volume 26, pages 46–49. Wiley Online Library, 2007.

[15] Sariel Har-Peled. Quadtrees—hierarchical grids. *Lecture notes*, 2010.

[16] Autodesk Help. Two-manifold vs. non-manifold polygonal geometry. `https://knowledge.autodesk.com/support/maya-lt/learn-explore/caas/CloudHelp/cloudhelp/2015/ENU/MayaLT/files/Polygons-overview-Twomanifold-vs--nonmanifold-polygonal-geometry-htm.html`, 2015. [Online; accessed 9-June-2018].

[17] Karsten Hilbert and Guido Brunnett. A hybrid lod based rendering approach for dynamic scenes. In *Computer Graphics International, 2004. Proceedings*, pages 274–277, Piscataway, NJ, US, 2004. IEEE.

[18] Christof Koch and Shimon Ullman. Selecting one among the many: A simple network implementing shifts in selective visual attention. Technical report, Massachusetts Institute of Technology Computer Science & Artifical Intelligence Lab, Cambridge, MA, USA, 1984.

[19] Kok-Lim Low and Tiow-Seng Tan. Model simplification using vertex-clustering. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 75–ff, New York, NY, USA, 1997. ACM.

[20] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 199–208, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[21] David Luebke and Benjamin Hallen. Perceptually driven interactive rendering. *University of Virginia Tech Report CS-2001-01*, 2001.

[22] David P Luebke. *Level of detail for 3D graphics*. Morgan Kaufmann, Burlington, MA, USA, 2003.

[23] Jun Miao, Feng Huang, Sreenath Narayan, and David L Wilson. A new perceptual difference model for diagnostically relevant quantitative image quality evaluation: a preliminary study. *Magnetic resonance imaging*, 31(4):596–603, 2013.

[24] Jun Miao, Donglai Huo, and David L Wilson. Quantitative image quality evaluation of mr images using perceptual difference models. *Medical physics*, 35(6Part1):2541–2553, 2008.

[25] Deepa Naik. 3D Mesh Simplification Techniques for Enhanced Image Based Rendering. Master thesis, Tampere University of Technology, Department of Science and Technology, 2017.

[26] Kenneth H Rosen. Discrete mathematics and its applications. *Amc*, 10(12):824, 2007.

[27] Daniel Scherzer and Michael Wimmer. Frame sequential interpolation for discrete level-of-detail rendering. In *Computer Graphics Forum*, volume 27, pages 1175–1181. Wiley Online Library, 2008.

[28] D Amnon Silverstein and Joyce E Farrell. The relationship between image fidelity and image quality. In *Image Processing, 1996. Proceedings., International Conference on*, volume 1, pages 881–884, Piscataway, NJ, US, 1996. IEEE.

[29] Veronica Sundstedt, Alan Chalmers, Kirsten Cater, and Kurt Debattista. Top-down visual attention for efficient rendering of task related scenes. In *VMV*, volume 4, pages 209–216, 2004.

[30] Michal Valient. Taking killzone shadow fall image quality into the next generation. *Presentation at GDC*, 14, 2014.

[31] Rui Wang, Bowen Yu, Julio Marco, Tianlei Hu, Diego Gutierrez, and Hujun Bao. Real-time rendering on a power budget. *ACM Transactions on Graphics (TOG)*, 35(4):111, 2016.

[32] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[33] Hector Yee. Perceptual metric for production testing. *Journal of Graphics Tools*, 9(4):33–40, 2004.

[34] Hector Yee, Sumanita Pattanaik, and Donald P Greenberg. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. *ACM Transactions on Graphics (TOG)*, 20(1):39–65, 2001.

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden