UNIVERSITÀ DEGLI STUDI DI PADOVA

Department of Information Engineering

Master's thesis in Computer Engineering

DEEP 3D MODEL OPTIMISATION FOR IMMERSIVE INTERACTIVE APPLICATIONS

ACADEMIC YEAR 2020-2021

6 DECEMBER 2021

Advisor:

Prof. SIMONE MILANI

Co-advisor:

Prof. FEDERICA BATTISTI

Author:

FRANCESCO PHAM

Dedicated to my family

Abstract

In recent years, 3D applications and immersive technologies reformed many industries including entertainment and health.

In this work, we want to present some possible applications of Deep Learning in 3D optimisation with a focus on immersive applications such as Virtual Reality and Mixed Reality. Our objective is to improve the Quality-of-Experience in such resource-demanding applications by selecting the parameters that result in a good balance between different quality issues.

To achieve our goals, in this work we examine the current state-of-the-art in quality metrics, how they can be applied to our domain, and how Deep Learning can assist in providing such metric values.

Sommario

Negli ultimi anni, le applicazioni 3D e le tecnologie immersive hanno riformato molti settori, tra cui l'intrattenimento e la medicina.

In questo lavoro di tesi, vogliamo presentare alcune possibili applicazioni del Deep Learning nell'ottimizzazione 3D prestando particolare attenzione su applicazioni immersive come la Realtà Virtuale e la Realtà Mista. Il nostro obiettivo è migliorare la qualità dell'esperienza in tali applicazioni selezionando i parametri che determinano un buon equilibrio tra i diversi fattori che impattano qualità.

Per raggiungere i nostri obiettivi, in questo lavoro esaminiamo l'attuale stato dell'arte nelle metriche di qualità, come possono essere applicate al nostro dominio e come il Deep Learning può aiutare a fornire tali valori.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, online multimedia applications have become increasingly popular with the growing necessity of distance learning, remote working, and telehealth. These applications are made possible by the substantial increment of computational power in graphics hardware, together with the development of technical solutions for the 3D reconstruction and visualisation.

Even though the technology is improving rapidly, the computational demand of the applications is also very high. For instance, 4K video streaming involves a vast amount of data being transferred through the network; highly detailed and realistic video game rendering depends upon large graphics processing resources.

Many multimedia applications use 3D graphics, including Virtual Reality that applies 3D computer graphics to create a simulation of a virtual immersive environment. Immersive technologies are primarily used for entertainment and they also allow an interconnection of different users at diverse locations in an extended reality environment.

QoE issues There are multiple challenges that we face when trying to design an immersive application. Such systems usually run on a platform with limited resources such as mobile phones or Virtual Reality headsets. Online applications face the additional constraint of the limited bandwidth. Furthermore, there is an increasing demand for higher quality of service and 3D graphics software often requires a substantial amount of computational resources.

It can be seen that optimisation is the key for the best Quality-of-Experience (QoE) in such applications. Good quality metrics are essential in solving this optimisation problem, therefore, this thesis focuses on the main issues that affect the QoE.

Deep Learning is a powerful tool in this kind of optimisation problems: its effectiveness in learning from given data and in predicting output values makes it suitable for predicting QoE metrics.

In this thesis work, we examine the state-of-the-art in quality metrics as well as propose a deep learning approach for 3D graphics QoE optimisation, with particular focus on immersive applications.

1.1 Thesis outline

The rest of this thesis is organised as follows: Chapter 2 gives an introduction to immersive technologies by presenting some forms of immersive applications and an overview of the technologies supporting them; we also provide a brief introduction to 3D computer graphics. Chapter 3 inspects the state-of-the-art in QoE optimisation by stating an analogy in video streaming, and showing some 3D optimisation algorithms and frameworks. Chapter 4 talks about the most relevant subjective and objective quality metrics and illustrates why such metrics evaluate the QoE. Chapter 5 contains an introduction to the

Deep Learning theory, while Chapter 6 shows how we acquired the data for training our deep learning model as well as the architecture that makes up our neural network model. Finally, Chapter 7 shows the results of our experiments followed by Chapter 8 that contains the conclusions and some possible avenues for future research on the topic.

Chapter 2

Immersive technologies and 3D graphics

2.1 Immersive technologies

An immersive technology is a digital simulation of an experience in which the user can interact via advanced human-computer interaction technologies. We currently can define three types of immersive technologies: Virtual Reality, Augmented Reality, and Mixed Reality. These technologies are actually part of a continuous scale called reality-virtuality continuum ranging between the completely virtual (virtuality) and the completely real (reality).

Figure 2.1: Reality–virtuality continuum

Virtual reality (VR) is a complete immersive digital world that completely replaces the reality. The experience is commonly created by means of

VR headsets and the system usually also include: a binaural audio system; positional and rotational real-time head tracking allowing six degrees of movement; interactive controllers with haptic feedback; and additional sensors for a complete immersion experience.

Figure 2.2: Oculus Quest VR headset

Augmented reality (AR) combines both real and virtual elements. It allows to overlay digital objects such as digital images and annotations onto the existing reality by using devices such as the screen of a smartphone. As the name suggests, in AR the real-world environment is enhanced by computer-generated information. Similarly to VR, the immersion experienced is augmented using visual, auditory, and haptic sensory information.

Mixed reality (MR) is a step further, taking augmented reality to another level. The digital objects are visually integrated in the real world, and additionally, both real and virtual elements interact with one another and the user is able to interact with virtual objects like they would in the real world. MR combines the best technologies from VR and AR to provide a new immersive experience that can create many new useful applications in: engineering, design modelling, training, education, and more. Some good examples of MR headsets are Microsoft Hololens or the Lenovo Explorer.

Figure 2.3: Microsoft Hololens 2

2.1.1 Applications

There are many use cases of immersive applications. Currently they are used in:

- **Education**: the previously mentioned technologies makes learning more engaging by allowing the interaction with the environment in an immersive experience.
- **Engineering**: real-time simulations of engineering processes.
- **Entertainment**: improving gaming experience as well as interactive experience in the big screen industry.
- **Health care**: medical training is greatly improved with mixed reality applications with the 3D interactive visualisation of the human anatomy.

2.1.2 Forms and technologies

Immersive applications take different forms differentiated by the hardware running it as well as the level of immersion and interaction.

In **projector-based** VR experience, an immersive room is created by projecting the images to the walls of the room by a system of projectors. Modeling of the real environment is fundamental in this VR modality. Examples of such applications are airplane simulation, driving simulation, and construction modeling.

The most popular form of immersive technology involves the usage of a **Head-Mounted Display** (HMD) which consists of two small high resolution monitors with a wide field of view that provide separate images for each eye for a stereoscopic visualisation of a 3D virtual world. The system is usually integrated with gyroscopic and accelerometer sensors for rotational head tracking, allowing three degrees of movement (roll, pitch, and yaw); some headsets also have additional sensors scanning the surrounding area for positional tracking, allowing a total of six degrees of movement. An HMD device could require an external computing and graphics rendering device, this type of headset device is called *tethered*. On the other hand we have *stand-alone* VR headsets that have integrated computing platforms inside the HMD.

A **Desktop-based** VR system involve displaying the 3D virtual content on a 3D monitor, which is a large screen capable of displaying stereoscopic images for example, using a polarisation system. The main limitation of this form of immersion is that there is no sense of peripheral vision, reducing the user's ability to sense what is happening around them.

2.2 3D computer graphics

In the last decades, 3D graphics technologies have seen a substantial improvement over the years. Such advancements are made possible by the great increment in computational power in graphics processing units (GPU) as it can be seen in figure 2.4.

Progress has been made not only from the hardware perspective but also in software technology: advanced algorithms such as shading and global illumination techniques allowed highly realistic 3D model rendering.

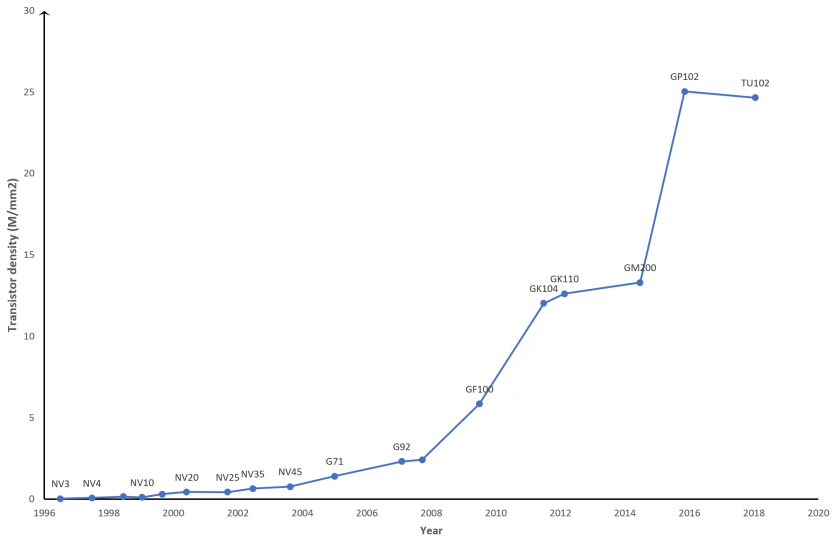

Figure 2.4: A chart of Nvidia's main GPUs showing changes in transistor density over the years [1]

3D models are represented in 3D computer graphics as a digital approximation of the shape and surfaces of the real physical body. This is achieved by making use of two main digital representations:

- **volume based**: the object is represented by defining its volume as a solid made of a certain material and composition. Volume data might

not consist of surfaces and edges at all. A volumetric object is typically represented as a large 3D grid of *voxels*.

- **boundary based**: the object's surfaces are described using a number of points in 3D space. These points are called vertices and they are interconnected by various polygons in a polygonal mesh. The constant memory requirements and guaranteed planarity of triangles make them preferable to generic polygons. Here we assume that these polygonal meshes are triangulated. Polygonal models currently dominate interactive 3D computer graphics, therefore this is the representation which we mainly focus on.

- **Point cloud**: A point cloud is a set of points in space. Each point is defined by its 3D coordinates and usually some additional properties are assigned to the point such as its colour. The point cloud often represents a 3D object by defining a set of points distributed on its surface or its volume.

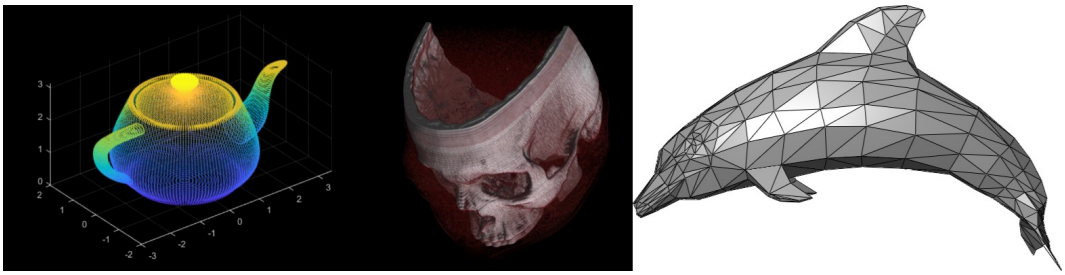

Figure 2.5: Illustrations of the three 3D representations. From left to right: point cloud, volume, and mesh

Detailed 3D models typically have a considerable number of triangles. For instance, Stanford University provides a repository of 3D laser scanned models

of physical objects. One of these models is the scan of a resin Thai statue, which consists of roughly 19 million vertices (39 million triangles).

Although rendering all triangles at once would result in detailed and realistic models, it would also include many details imperceptible to the human eye and the rendering would be notably slow, especially in interactive environments, impacting the QoE. In practice, mesh simplification and compression algorithms are used to reduce the number of indistinguishable details, saving a lot of computational resources that would otherwise be wasted and finally improving the QoE by reducing lag.

Chapter 3

The current state of the art

3.1 QoE optimisation in video streaming

There is an analogy between the problem we want to analyse and the issues encountered in video streaming services. Videos are normally compressed and reduced in resolution before transmitting to the client. Transmission errors, compression, and delays can all affect the QoE from the user's perspective.

The current state of the art in video transmission for improving the QoE employs adaptive video streaming, which is a technique to dynamically change the video bitrate, adapting to network conditions. An adaptive client can measure the available bandwidth and choose one of several video representations at different compression levels, keeping the best possible QoE while avoiding rebuffering events, i.e., emptying the playout buffer completely and freezing the video until a new segment is available.

Many frameworks have been proposed to perform bitrate adaptation, such as heuristic algorithms and Reinforcement Learning solutions. [13]

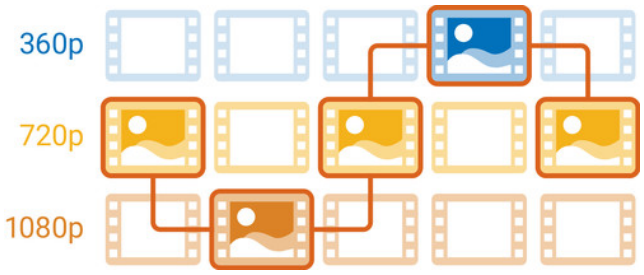

Figure 3.1: Illustration of adaptive bitrate streaming [2]

3.2 Mesh simplification

The triangle mesh decimation is an important tool in 3D rendering QoE optimisation. The goal of the decimation algorithm is to reduce the total number of triangles in a triangle mesh, while still maintaining the original topology and a good approximation to the original geometry. This allows us to remove the extra details that a human eye would not be able to discern.

The decimation algorithm is the following. Multiple passes are made over all vertices in the mesh. During each pass, all vertices that meet a specified criterion, and all adjacent edges and triangles are deleted. The resulting hole is triangulated. The vertex removal process repeats until some termination condition is met. Usually, the termination criterion is specified as a reduction in the percentage of displayed triangles with respect to the original mesh. [24]

Mesh simplification, however, does not come without downsides: an excessive mesh decimation would result in visible distortions, which would affect the 3D model fidelity. Hence, a good compromise between model complexity and quality is necessary to achieve a good QoE.

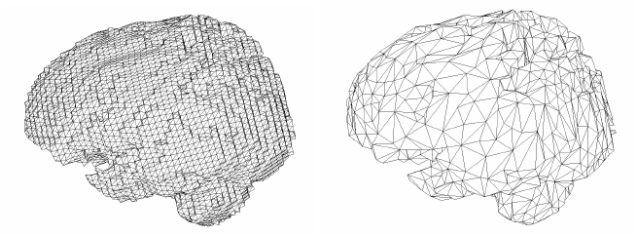

Figure 3.2: Example of decimation of triangles from a mesh of a brain [19]

3.3 Level of details

A good QoE would be achieved with an adequate trade-off between the quality of the rendered model perceived by the user and the refresh rate of the 3D model as the user moves.

However, estimating the number of triangles that would optimise both the amount of details in the 3D model and the frame rate is hard to obtain. The amount of detail that the user is capable of perceiving is also dependent on various factors such as the resolution of the display, the texture of the 3D model, the point of view of the user and the characteristics of the human visual system (HVS). In a MR environment, where the user is able to freely move around the object, we can assume that the display is fixed and, on the other hand, the position or the perspective of the user with respect to the object determines the details the viewer is able to distinguish. To exemplify, we can see that the farther the viewer is from the object, the fewer details the viewer is able to discern.

This suggests that it is necessary to make use of multiple Levels of Details (LODs) of the same 3D object from which to choose the optimal one on the basis of the previous considerations. The LODs are generated by taking a 3D mesh and performing the decimation algorithm multiple times to produce different versions of the same 3D object, each with different triangle count and

hence different amount of details.

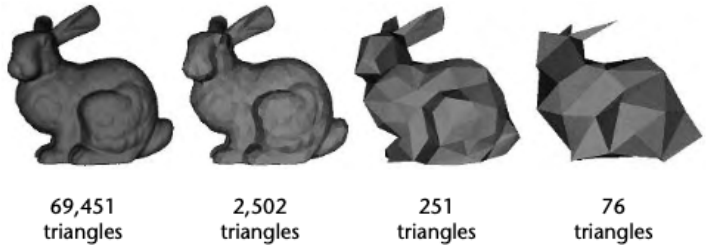

Figure 3.3: Multiple level-of-details of an object [23]

3.4 LOD selection factors

The most important issue we need to address when talking about LOD management is the selection of the proper LOD to render and when to switch to a lower or higher resolution model. Here we discuss the most common LOD selection frameworks.

Distance Simply assign to each LOD a range of distances at which it will be used to represent the object. This is the most simple way to manage the level of detail and it is based on the consideration that less details are visible for a distant object; as a consequence, we can select a lower LOD without greatly affecting the fidelity of the image. Despite its simplicity, distance LOD has some definite disadvantages. Choosing an arbitrary point within the object for all distance calculations introduces inevitable inaccuracy, since the actual distance to the viewpoint can change depending on orientation. This can lead to more obvious popping effects under certain conditions. [23]

Size Since objects appear smaller as they move further away, in an alternative to measuring the distance of the object in the world space, we can measure the size of the object as projected on the screen. Size-based LOD selection uses the projected screen coverage, of the object, and select a LOD based on a series of size thresholds instead of distances. Size-based techniques are more accurate compared to distance-based techniques. However, projecting the world coordinates of the object into screen coordinates can be computationally expensive. Many systems estimate the screen coverage of the object by projecting its bounding box into screen space.

Priority In many interactive environments, some objects are more important than others. Therefore, some systems implement schemes in which objects receive a priority ranking, so that those objects deemed most important are degraded the least.

Environmental conditions LOD thresholds can vary depending on various environmental conditions, such as fog, smoke, and clouds.

Perceptual factors LOD selection frameworks should take into consideration also our human visual system. For example, we can perceive less detail for objects in our peripheral vision, or for objects moving rapidly across our gaze. We could therefore imagine reducing the resolution of objects under these circumstances, improving the frame rate without perceptible degradation. [23]

Chapter 4

QoE metrics

As previously noted, in 3D model rendering, mesh simplification algorithms are often used for saving computational resources, thus enabling a seamless interaction with the 3D object in an immersive application. Such algorithms introduce some distortion and degradation to the visual quality of the 3D model.

Therefore, maximising the quality of experience for rendered content requires some compromises on quality parameters depending on the computational resources available. Yet, it is currently unclear which parameters best maximise perceived quality. This is an optimisation problem that requires a good perceptual quality metric to have a good LOD selection framework.

4.1 Subjective quality

Since computer graphics applications, after all, are viewed by human beings, the only "correct" method of quantifying perceptual quality is through subjective evaluation. The most immediate subjective quality metric is *Mean Opinion Score* (MOS): it is simply the average of a series of users' evaluations, going

from 1 to 5, in controlled experiments.

However, in computer graphics and interactive applications, we need to evaluate the quality in real time and for obvious reasons, MOS is not practical. Therefore, we need to assess quantitative measures that can automatically predict perceived quality, i.e., an objective quality index. Here we take advantage of a *Mesh Visual Quality* (MVQ) metric inherited from a *Image Quality Assessment* (IQA) metric.

Subjective quality evaluations are useful for comparing the results with objective metrics, as well as to determine the main factors that affect the QoE. Subjective data are essential to design reliable objective quality indices that accurately assess perceived quality.

4.2 Image Quality Assessment metrics

One of the most used image quality metrics is the *Peak signal-to-noise ratio* (PSNR). PSNR is derived from the mean square error and indicates the ratio of the maximum pixel intensity to the power of the distortion.

$$PSNR_{dB} = 20 \cdot log_{10}(MAX_i) - 10 \cdot log_{10}(MSE)$$

Another important metric is the *Structural Similarity Index Measure* (SSIM), which is used for measuring the similarity between two images and is used to assess the perceived quality in 2D images and videos using known properties of the human visual system. The SSIM metric combines local image structure, luminance, and contrast into a single local quality score. In this metric, structures are patterns of pixel intensities, especially among neighbouring pixels, after normalising for luminance and contrast. The SSIM between two windows x and y of the frame X with respect to a reference frame Y is given by:

$$SSIM = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

where μ and σ are the local mean and standard deviation of the two images, C_1 and C_2 are constants used to stabilise the equation in the presence of weak denominators, and σ_{ij} is the covariance. The range of SSIM values extends between -1 and +1 and only equals 1 if the two images are identical. It is conventional for SSIM to be calculated using a sliding window, typically of size 11×11, using a circularly symmetric Gaussian weighting function. The SSIM value for the whole image is then computed as the average across all individual windowed results. [25]

Both the PSNR and the SSIM index are full reference metrics; that is, the measurement or prediction of image quality is based on an initial uncompressed or distortion-free image as reference. Because the human visual system is good at perceiving structure, the SSIM quality metric agrees more closely with the subjective quality score, therefore, it is the chosen metric for this project.

4.3 Mesh Visual Quality

Two categories of Mesh Visual Quality (MVQ) metrics can be defined: **image-based** and **model-based**. In this thesis work we focus on the former category.

An image-based MVQ metric makes use of 2D image quality metrics, this decision is supported by the following considerations: a 3D environment is ultimately viewed by the user as a sequence of 2D frames, including stereoscopic images which are composed of pairs of 2D frames. Therefore, by means of a 2D perceptual metric we can estimate the mesh perceptual quality from the observer's viewpoint. For instance, in this thesis project, the best level-of-detail is used for reference and it is compared with a lower LOD using the SSIM

index.

Model-based MVQ metrics, on the other hand, are derived from the geometry of the 3D model, therefore they are independent from the observer's viewpoint. However, such metrics usually are only restricted to the geometric information without taking into consideration the 3D model's texture and shadows, unlike image-based metrics.

4.4 Frame rate

The MVQ metric evaluates the QoE only in a static context, since it measures just the visual quality of a single frame without taking into consideration the responsiveness of the system. The temporal aspect, that is, how the sequence of frames is distributed in time, is also as important as the MVQ.

In an interactive 3D environment the second most important factor which affects the QoE, in addition to the MVQ, is the frame rate. Sometimes known as "update rate" or "refresh rate," the frame rate is the frequency in which a consecutive series of frames are displayed. The most common frame rates used in television and movies are between 24 to 60 fps. The temporal sensitivity of the human vision varies depending on the type and characteristics of the visual stimulus, and it differs between individuals. Some people can perceive the flicker in a 50 or 60 Hz light source, and at higher refresh rates perceptible flicker is reduced and the modulated light is perceived as steady. [18]

Several studies have shown that lag or extreme variance in frame rate can impact the user experience and, in an extended reality environment, can cause nausea and motion sickness [12]. Therefore, we can clearly see that this is an important metric in determining the QoE.

4.5 QoE Deep Learning prediction

Sometimes it is advantageous to have a prediction of the QoE metrics, especially in LOD management. In LOD selection we need to select a LOD that best optimise the QoE of the successive frames that aren't rendered on screen yet. Moreover, the SSIM index is computationally expensive to compute and it is not practical to compute it in a real-time interactive environment in any case. Therefore, we need to predict the QoE in order to evaluate how each LOD would perform in that particular situation, thus select the optimal one.

In this project we study the prediction of QoE metrics by the means of Deep Learning techniques. We specifically try to predict the mesh perceptual quality and the frame rate given the viewpoint position of the observer and some additional information on the 3D mesh.

Chapter 5

Deep Learning Theory

In this chapter we are going to give a brief survey and some definitions of the concepts and techniques of machine learning that will help us develop our approximation of a function assessing the QoE metrics we have discussed.

5.1 Introduction to Machine Learning

Machine Learning is a sub field of Artificial Intelligence. The goal of Machine Learning is to design computer algorithms that make machines that are capable of learning from a training data and executing tasks without being explicitly programmed to perform. Therefore, a Machine Learning model is a mathematical model learned directly from the training data, without predetermined algorithms and mathematical equations as in classical algorithms.

Definition 5.1. A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

Tom M. Mitchell

The training data represents the past experiences used by the machine to learn and improve future performance on a certain task. The performances improve as the training data become larger. Our data set will be discussed in detail in chapter 6.

The tasks that Machine Learning aims to resolve are classified into three main categories:

- **Supervised learning**: Each example of the training set includes the inputs and the desired outputs. An example set is presented to the machine which aims to approximate the relationship between the inputs and outputs. The final objective is therefore to train a model capable of predicting the desired outputs in a set of examples that have never been presented to the algorithm.

- **Unsupervised learning**: In contrast to supervised learning, the goal of an unsupervised learning task is to extract patterns from the data in which the outputs corresponding to the inputs are unknown. The machine is provided instead with a copious amount of data from which the algorithm has to infer hidden characteristics and structures.

- **Reinforcement learning**: It is based on the assumption that the machine is able to receive some external feedback of the action executed by the algorithm coming from a dynamic environment. In opposition to supervised learning in which the correct actions the system has to execute are provided, in reinforcement learning only qualitative information on how effectively the machine is performing are available.

5.1.1 Supervised Learning

The problem faced in this thesis is a supervised learning task, therefore in this discussion we mainly focus on this approach.
As discussed previously, Supervised Learning is a machine learning approach that requires a dataset. This data is split into three independent sets:

- **Training set**: The sample of data used to fit the model.
- **Validation set**: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.
- **Test set**: The sample of data used to provide an unbiased evaluation of a final model fit to the training dataset.

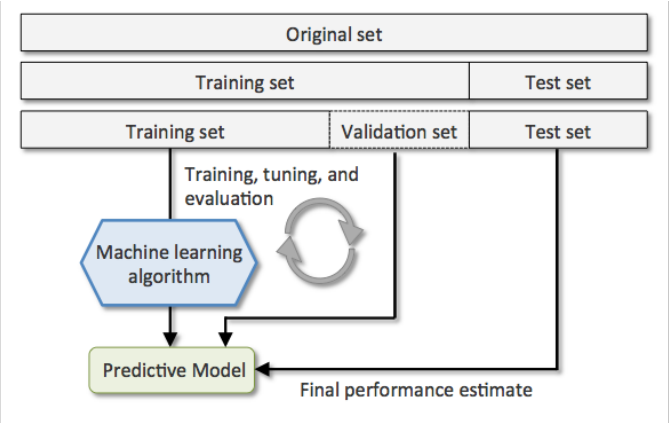

Figure 5.1: Data set split into training - validation - test subsets [4]

5.2 Artificial Neural Networks

There exist numerous techniques for designing and implementing a machine learning model: from decision trees to genetic algorithms, just to mention

a few. The most popular technique and powerful one is based on Artificial Neural Networks.

An *Artificial Neural Network* (ANN) is a mathematical model inspired by biological neural networks as the "neural" part of the name suggests. ANN is the subject of numerous studies and its importance derives from its applications in many different fields. It is composed of a number of processing units, called neurons, and the connections between them imitate the functions of biological synapses.

5.2.1 Perceptron

The simplest ANN model is the *Perceptron*. Introduced by Rosenblatt in 1958, the Perceptron is the simplest approximation of the biological neuron's characteristics. The biological neurons take input from various other neurons through the dendrites, perform the required processing on the input and send another electrical signal through the axon to the terminal nodes where the signal is propagated to numerous other neurons. The Perceptron emulates the same behaviour in a simple mathematical model.

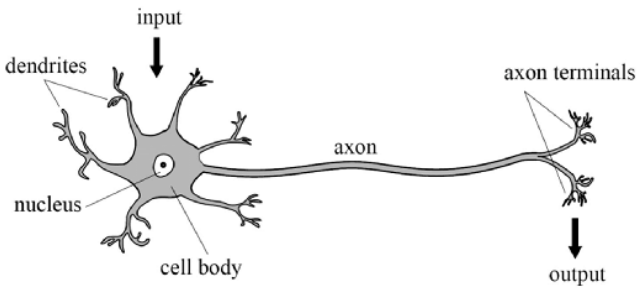

Figure 5.2: Biological neuron [10]

A perceptron takes a vector of real-valued inputs on which it calculates a linear combination. The result is forwarded to a function, called *activation*

function which returns the output of the neuron. The computation can be summarised in the following mathematical equation:

$$y = \sigma\left(\sum_{i}^{n} w_i x_i + b\right) \tag{5.1}$$

where $(x_1, .., x_i, .., x_n)$ is the input vector, each w_i is a real-valued weight, that determines the contribution of input x_i to the output and b is a constant called bias.

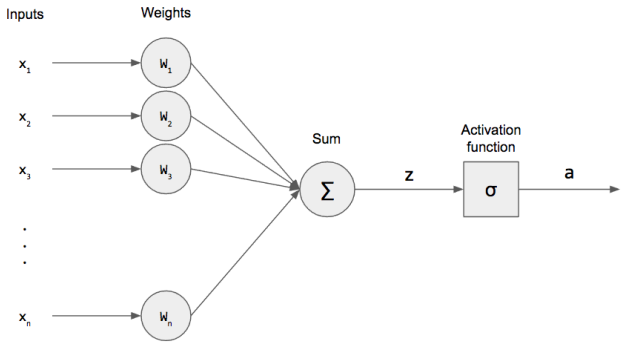

Figure 5.3: Perceptron [16]

In the classical perceptron, the activation function σ is a step function:

$$\sigma(z) = \begin{cases} 1 & \text{if } z > 0. \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

However, many other alternative activation functions are widely used, the most popular ones are:

- ***Sigmoid***: $\sigma(z) = \frac{1}{1+e^{-z}}$
- ***Rectified Linear Unit (ReLU)***: $\sigma(z) = max(0, x)$

5.2.2 Deep Neural Network

A *Deep Neural Network (DNN)* is an extension of the main concepts behind the Perceptron. Multiple layers of neurons are stacked into a multilayer network. We can distinguish the *input* and *output* layer, and the layers between them are called *hidden layers*.

The *Feedforward Neural Network* is a DNN in which the signal only propagates in one direction, from the input layer to the output layer, and the connections between the nodes do not form a cycle.

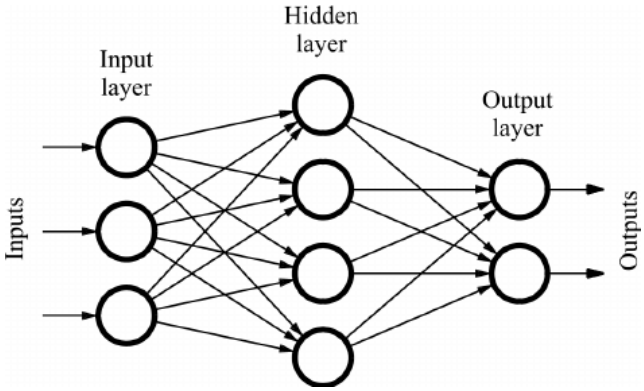

Figure 5.4: Feedforward Neural Network [17]

5.3 Backpropagation

Until now, we have discussed the basic structure of an ANN. Although the architecture is important in defining the ANN, the "learning" part of *Deep Learning* is also equally essential. In order to teach something to the model we need to introduce *Backpropagation*. This algorithm allows the network to modify its weights and biases such that we can obtain the desired outputs given certain inputs.

When we use a feedforward neural network to accept an input x and produce an output $\hat{y}$, information flows forward through the network. The input x provides the initial information that then propagates up to the hidden units at each layer and finally produces $\hat{y}$. This is called forward propagation. The performances of the model are then evaluated with a loss function. The backpropagation algorithm, allows the information from the cost to flow backward through the network to compute the gradient.[11]

5.3.1 Loss function

A *loss function* (or "cost" function) is used to quantify the output error of the network. A common loss function is the *Mean Squared Error* (MSE) and is given by the following equation:

$$\mathcal{L}(\Theta) = \frac{\sum_{i=1}^{T}(y_i - \hat{y}_i)^2}{T} \tag{5.3}$$

where $Y = (y_1, y_2, ..., y_T)$ is the vector of true values (the "correct answers"), $\hat{Y} = (\hat{y_1}, \hat{y_2}, ..., \hat{y_T})$ is the vector of model predictions, T is the number of samples and Θ is the current network model.

Other loss functions are also used. For instance, the *Mean Absolute Error* (MAE) which is the sum of absolute differences between our target and predicted values:

$$\mathcal{L}(\Theta) = \frac{\sum_{i=1}^{T}|y_i - \hat{y}_i|}{T} \tag{5.4}$$

In short, MSE penalises the outliers in the dataset more, at the expense of common examples. Therefore, MAE is more robust if the training data is corrupted with outliers.

5.3.2 Gradient Descent

Backpropagation is based on the classical concept of derivative to identify the minimum of the cost function. Since in machine learning we deal with multivariate functions, we must make use of the gradient. The gradient is an operator used in vector calculus which points in the direction of the steepest increase of a function. If we use it with a negative sign, we can find the direction of fastest decrease of the cost function. Therefore, the main idea of the Gradient Descent rule is to update the weights Θ with the following rule:

$$\Theta = \Theta - \eta \frac{\partial \mathcal{L}}{\partial \Theta} \tag{5.5}$$

where η is the *learning rate* which is a parameter that defines the step size while moving towards the minimum of the loss function. A too high learning rate will make the learning jump over minima, but a too low learning rate will either take too long to converge or get stuck in an undesirable local minimum.[20]

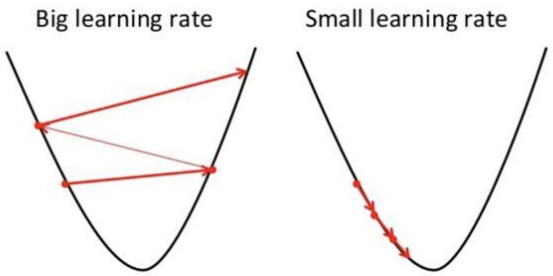

Figure 5.5: Learning rate [6]

5.3.3 Backpropagation

Backpropagation starts by calculating the partial derivatives of the loss function with respect to the weights of the last layer, which don't influence over

any other network parameters. This is calculated by applying the chain rule to the neuron's equation 5.1.

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_j} \cdot \frac{\partial a_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ij}}$$

Where a_j is the output of neuron j, and z_j is the net input signal before activation.

In the last factor of the right-hand side of the above, only one term in the sum z_j depends on w_{ij}, so that

$$\frac{\partial z_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}(\sum_{i}^{n} w_i x_i + b) = \frac{\partial}{\partial w_{ij}} w_{ij} x_i = x_i$$

The derivative of the output of the neuron with respect to its net input is simply the partial derivative of the activation function:

$$\frac{\partial a_j}{\partial z_j} = \frac{\partial \sigma(z_j)}{z_j}$$

The first factor is straightforward to evaluate if the neuron is in the output layer:

$$\frac{\partial C}{\partial a_j} = \frac{\partial C}{\partial y} = \frac{\partial}{\partial y} \frac{1}{2}(y - \hat{y})^2 = y - \hat{y}$$

However, if j is an arbitrary inner layer, finding the derivative of C with respect to a_j is less obvious. The derivative with respect to a_j can be calculated, thanks again for the chain rule, if all the derivatives with respect to the outputs of the next layer are known. And so, after we have computed the partial derivatives with respect to the weights of the last layer, we'll continue progressing backwards, until we reach the beginning of the network.

We can see that, by iterating backward we avoid redundant calculations of intermediate terms in the chain rule. Backpropagation allows an efficient computation of the gradient, which we can use to apply Gradient Descent and update all network parameters such that the cost function is minimised.

Finally, these steps are repeated over the entire training set until a certain termination condition is reached.

5.4 Convolutional Neural Network

The neural networks so far considered consist of neurons fully interconnected, that is, all neurons are connected with each neuron of the previous and the subsequent layers. A fully connected network has some disadvantages: the amount of weight to manage grows very quickly with the increase of the neurons per layer and the number of layers, this results in considerable memory requirements and training time. The large number of parameters also makes the network more susceptible to *overfitting*. This means that a network trained on a training set which is not large enough would perfectly analyse the training data, but would perform poorly on future observations.

These problems are effectively tackled by *Convolutional Neural Networks* (CNN). In contrast to traditional neural networks, the neurons in a CNN are connected to only a subset of neurons of the preceding layer, in a region called *receptive field*. This is called sparse connectivity, in contrast to a dense connectivity of a typical neural network. The CNN takes inspiration from the visual cortex of our brain. The local connection of this neural architecture is particularly suited for analysing images, since they usually have very high correlation between near pixels and low correlation between far points.

5.4.1 Convolution

In Convolutional Neural Networks, the response of a neuron to its receptive field is approximated by means of the *convolution* operator. In mathematics,

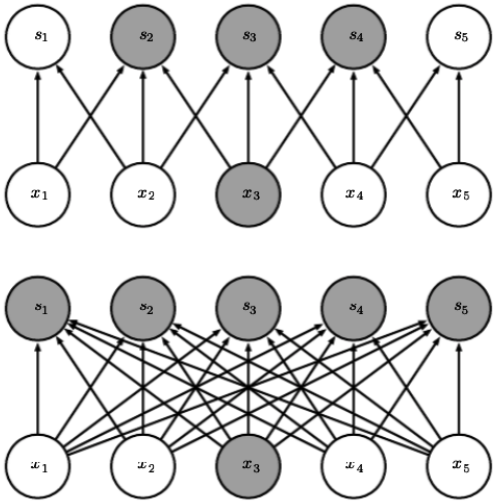

Figure 5.6: (Top) Sparse connection due to small convolution kernel. (Bottom) Dense connections. [11]

the convolution is an operation applied to two real continuous functions:

$$(x * k)(t) := \int_{-\infty}^{\infty} x(\tau)k(t-\tau)d\tau \tag{5.6}$$

In a CNN x is the multidimensional input and k is a multidimensional *kernel* containing the parameters which are updated by the training algorithm. In the case of bi-dimensional images the discrete convolution operation becomes the following:

$$(I * W)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{5.7}$$

The output of the convolution is called *feature map*.

The characteristics of this operator allow us to maintain a limited number of parameters and provide the useful property of translation invariance. In other words, the same kernel is applied over all the input image in a sliding window approach, which allows to detect a feature in the image independently from its position.

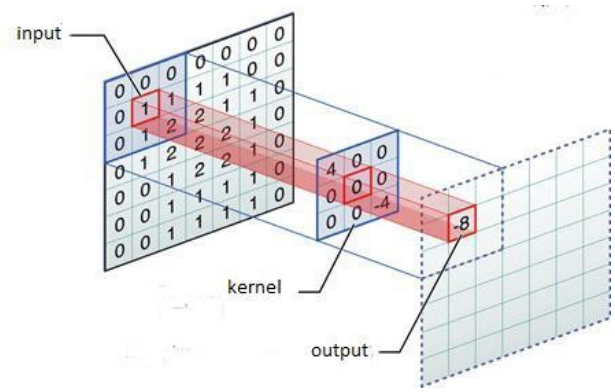

Figure 5.7: Illustration of the convolution operation [14]

5.4.2 Activation function

The convolution operation is usually immediately followed by an activation layer, similarly to traditional neural networks. It enables the model to learn non-linear functions. The most common activation function used in recent years is the *Rectifier Linear Unit* (ReLU), which is a simple yet effective activation operator.

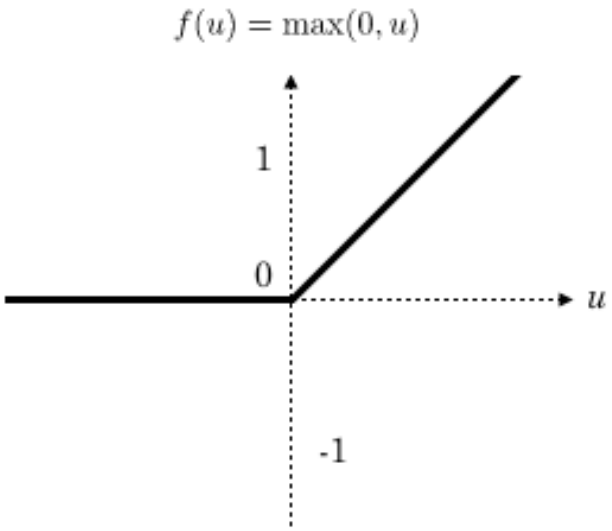

Figure 5.8: ReLU activation function

5.4.3 Pooling layer

The pooling layer is often placed in between successive convolution layers in a CNN architecture. Its function is to progressively reduce the number of parameters, and thus to also minimise overfitting. The most common form is a MAX pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice of the input by 2 along both width and height, discarding 75% of the activations. [3]

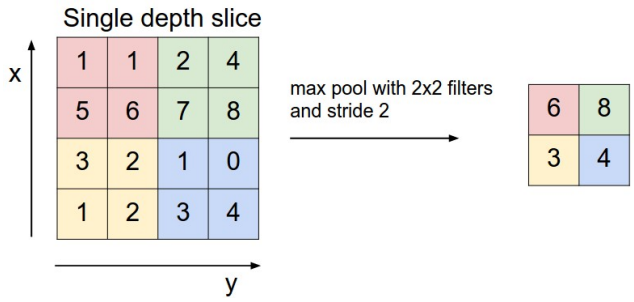

Figure 5.9: Illustration of the pooling operation [3]

Chapter 6

Dataset and neural network architecture

As we have mentioned, QoE optimisation requires good quality metrics. Here we will be focusing on objective quality metrics, and in future work we will consider subjective data for further fine-tuning the model and evaluate the effectiveness of objective metrics in predicting perceived quality. The objective quality indices that we mostly look into in this thesis project are the SSIM index and the frame rate. Considering the issues we have previously mentioned, our goal is to have a good Deep Learning model which is able to provide a good prediction of these quality indices in real time.

Unity In this thesis project, we acquired data using the Unity game development platform. Unity is developed by *Unity Technologies* and provides a cross-platform game engine which supports a variety of desktop, mobile, console and virtual reality platforms. Unity started as a game engine with basic graphics capabilities and physics calculations and quickly grew into a well known tool among developers with more than 6 million registered developers.

Furthermore, it is often picked as the development platform of choice for extended reality applications. [9]

Unity provides a user-friendly Integrated Development Environment (IDE) and it is frequently used to create 3D, 2D, VR, and AR interactive visual content and video games. In Unity, we can easily: import 3D models including its textures and materials; add a camera which defines a view in the scene space; add light sources that allow Unity engine to compute the global illumination and the shading of the Game Objects. Most importantly for our project, Unity also provides scripting capabilities in the C# programming language that will allow us to: move the camera to some position; take screenshots for generating our dataset; log some rendering statistics such as FPS and vertex count.

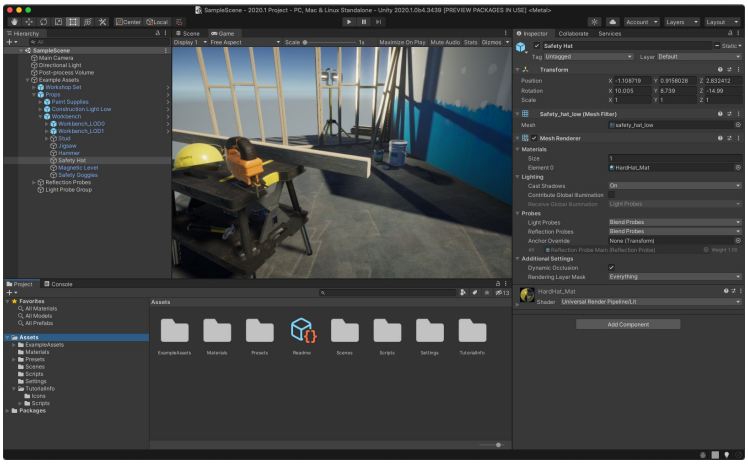

Figure 6.1: Unity game development platform [8]

6.1 Data acquisition

3D models First of all, we picked nine 3D models to use for training and test set generation. In figure 6.2 we can see the thumbnails of these 3D objects. A number of these models are picked from the Stanford 3D Scanning Repos-

itory, others have been downloaded from online 3D model platforms such as Sketchfab, Turbosquid, and Cgtrader. Eight of these models will be used to generate the training and validation set, while the "mercedes-benz" model is reserved for the test set.

LODs generation Secondly, we have generated multiple LODs for the 3D models we wanted to analyse. This was done using the "decimate modifier" tool of *Blender*, a free and open-source 3D computer graphics software toolset. We applied mesh decimation multiple times in order to produce four different LODs for each 3D model. For instance, each LOD of Lacock Abbey 3D model in figure 6.3 have the following vertex count: 3.5M, 1.3M, 504k, and 211k.

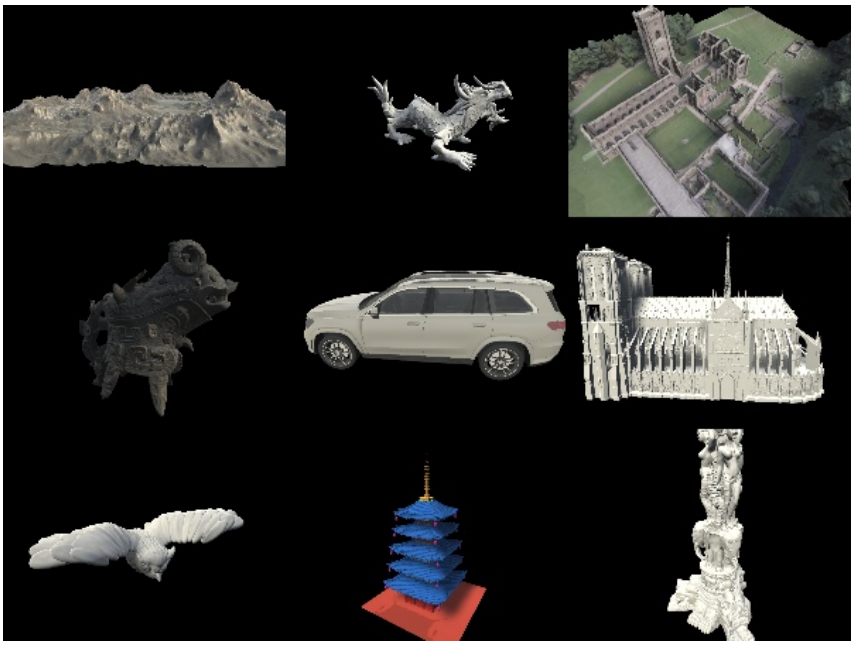

Figure 6.2: The 3D models used for generating training and test set

Viewpoint generation Seeing that the quality indices are dependent on the observer's viewpoint, the first step is to generate a set of random positions

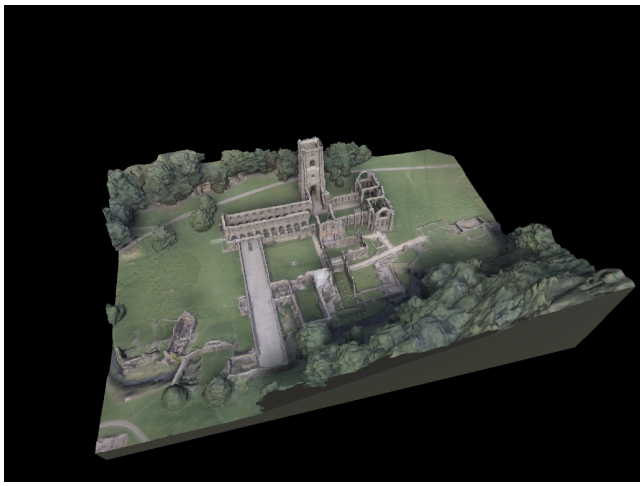

Figure 6.3: Lacock Abbey 3D model

around the 3D object. The positions are generated in a uniformly manner, this way we have a uniform sampling of the quality and complexity of the 3D model viewed from any direction. These positions are then fed into a Unity script which takes a screenshot in every generated camera position and every LOD of the 3D object. Here we are considering a camera which is always looking at the centre of the 3D object, in future work we may generalise the "look at" vector of the camera.

6.2 SSIM index

6.2.1 Static SSIM

As previously noted the SSIM index can be used to evaluate the perceived quality of a 3D model. In a first experiment, we have determined the SSIM quality index at each camera position and for each LOD. Here, both frames being compared are taken from the same camera position, therefore we called it *static SSIM*. We will later see that comparing frames taken from different

camera positions can provide additional information to the user's movements.

Let us consider the 3D model of Lacock Abbey, from which we have generated four LODs. We use the best version as the reference and given a specific camera position, we compare the reference LOD with the other LODs of the mesh, resulting in three SSIMs for each camera position. As we can see in figure 6.4, the SSIMs are typically higher as the camera is positioned farther from the object.

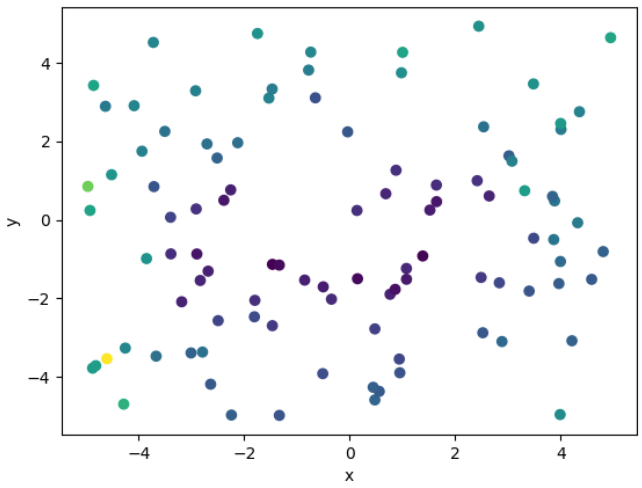

Figure 6.4: SSIMs at different camera positions. The data point colour represents the SSIM score.

The correlation is more visible in figure 6.5 which shows the distance of the camera from the center with respect to its SSIM. We can also see that less detailed LOD have lower SSIMs as expected.

6.2.2 Dynamic SSIM

The QoE in an extended reality environment is affected not only by the quality of the 3D model but also by the lag in the animation. The lag depends on

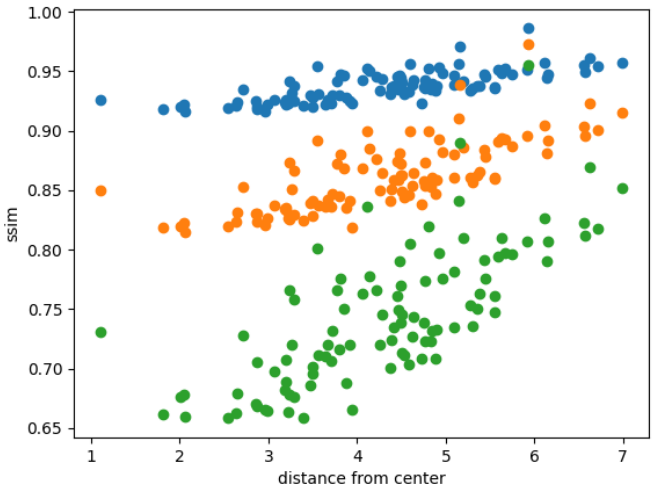

Figure 6.5: Scatter plot showing that the farther the observer is from the object, the closer to one the SSIM index is. Each LOD is represented by a different data point colour.

two main factors: the frame rate of the rendering and the speed of the camera as the user moves around the rendered mesh. To exemplify, when the camera is still, a low frame rate is not noticeable, whereas it is perceived more easily when the observer is quickly moving around the object. However, it is incorrect to directly use the camera speed since a camera movement farther away from the 3D model matters less than a closer one at the same speed.

In our approach, we project the camera position a constant delta time into the future, e.g., t=0.1s, and we calculate the difference in visual appearance between the frame in the current position and the one projected into the future, using the SSIM index. We call this *dynamic SSIM* to distinguish it from the previously introduced static SSIM. The faster the user is moving, the farther apart are the camera positions, and the lower the SSIM is. In order to include the evaluation of the visual quality of the target LOD, we compare the frame

in the current position using the reference LOD and the frame in the projected position using the target LOD.

In short, the dynamic SSIM is computed by comparing two frames taken from a pair of camera positions. Therefore, the question is: given any pair of camera position, is it possible to predict the SSIM perceived similarity index? As we will see, for the purpose of answering this question, we trained a deep learning model to perform this task with good accuracy.

6.3 Frame vertex count

We have seen that one important factor affecting the QoE is the frame rate. The update rate, however, could be an unstable measurement that depends on the performance of the device. Its fluctuation and device-dependent nature makes its prediction more challenging.

In 3D graphics rendering, it is intuitive that the larger the number of vertices to be rendered is, the longer it takes to render a single frame (and therefore, the refresh rate is going to be lower). The vertex count of a frame can be easily retrieved using the following command:

```
UnityEditor.UnityStats.vertices
```

This number is not to be confused with the total number of vertices of the 3D mesh. The frame vertex count is in fact the number of vertices required for rendering single frame from a specific viewpoint. Moreover, texture mapping and normals can also increase the number of vertices computed in a frame.

In figure 6.6 we can clearly see the correlation between the frame rate and the vertex count. Below a certain vertex count, the Fps is limited to a range around 60 fps because the refresh rate of my display is 60Hz so the fps is capped to that frequency.

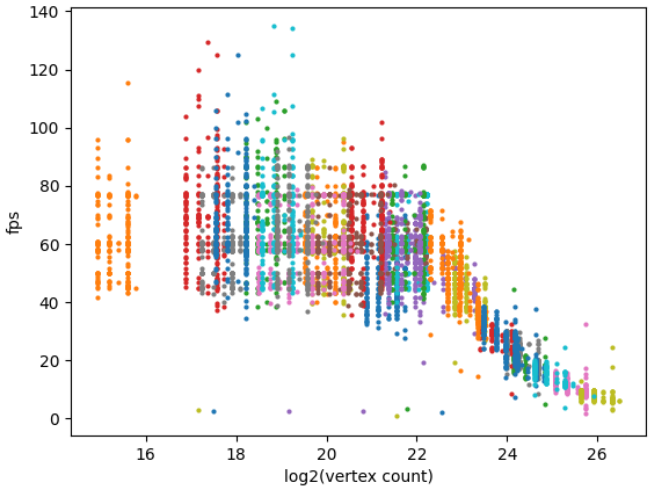

Figure 6.6: Scatter plot showing for each frame in our data set, its vertex count in logarithmic scale on the x axis, and its frame rate on the y axis.

Given this result, instead of predicting the fps which is unstable and depends on the performance of the device, we can predict instead the number of vertices to be rendered which is independent from the rendering device.

6.4 Orthographic triangle count projections

Firstly, we did some experiments training a neural network on a single 3D model. After having observed that it is possible to train a model to accurately predict the SSIM index, we decided to train a generalised model that it is able to handle multiple 3D objects with the purpose of generalising also to 3D models not present in the training set. To achieve this, it is necessary to provide the neural network some information on the 3D mesh structures.

The approach we have chosen consists of computing three orthographic projections of the 3D mesh, where instead of projecting the colours we projec-

ted triangle counts. The projection planes we have chosen are parallel to the principal axes. Each pixel of a view in the projections is computed by casting a ray and counting the number of intersecting triangles. These projections will be used by a CNN to extract some features of the mesh structure that will be useful for predicting quality indices of a generic 3D model.

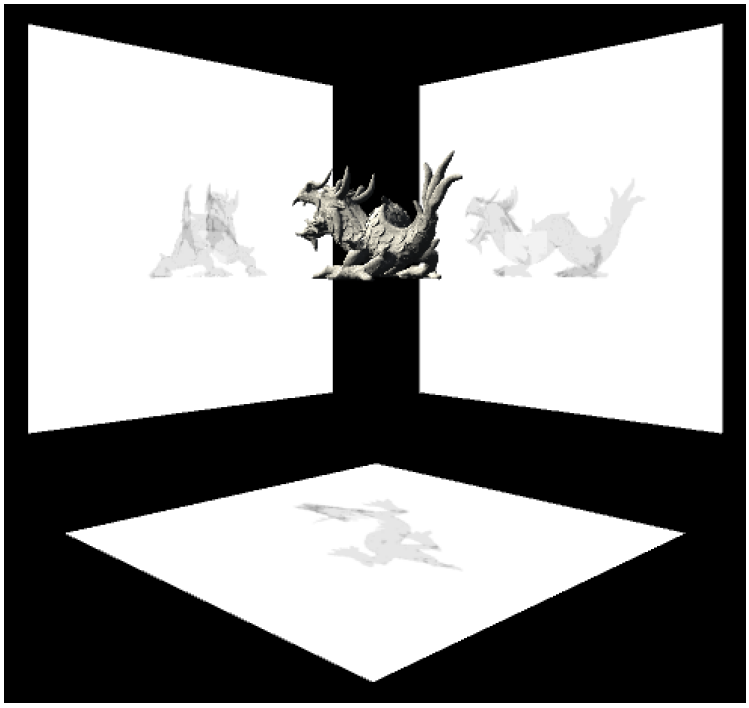

Figure 6.7: Illustration of the projection of the "Asian Dragon" 3D model onto the three views.

6.5 Dataset

Our final dataset consists of 10k samples generated from 8 different 3D models. Each sample include:

- Reference position: a 3D reference camera position in Cartesian coordinates;

- Target position: a 3D target camera position in Cartesian coordinates;
- Total vertex count: the total number of vertices in the 3D mesh in the target LOD;
- Projections: the orthographic triangle count projections of the 3D object;
- SSIM score: the similarity score between the frame taken from the reference position at reference LOD and the frame captured at target position at target LOD;
- Frame vertex count: the vertex count for rendering the frame at target position and target LOD;

Each pair of positions are generated randomly and each coordinate is picked in a range between -1 and 1. The 3D model is scaled to fit a 1x1x1 box and centred around the origin. The position pairs are generated such that the euclidean distance between target and the reference position is less than or equal to 0.3. This max distance value is a flexible parameter and it's based on the assumption that in our interactive application, it's unlikely to have huge and sudden camera jumps.

The input of our neural network is composed of: reference and target position, total vertex count, and projections. Whereas the outputs it has to predict are SSIM score and frame vertex count.

$$\begin{cases} \text{Reference position} \\ \text{Target position} \\ \text{Tot vertex count} \\ \text{Projections} \end{cases} \mapsto \begin{cases} \text{SSIM score} \\ \text{Frame vertex count} \end{cases} \tag{6.1}$$

Training and validation set We have split the 10k samples to reserve 20% of the dataset for validation and the remaining 80% for actual training, resulting in 8k training samples and 2k validation samples.

Test set The test set is composed of an additional 1k samples generated from a 3D model which is not included in the training set. This test set will be used to evaluate the deep learning model effectiveness in generalisation with different and new 3D models.

6.6 Neural network architecture

In figure 6.8 we can see an illustration of the overall neural network architecture.

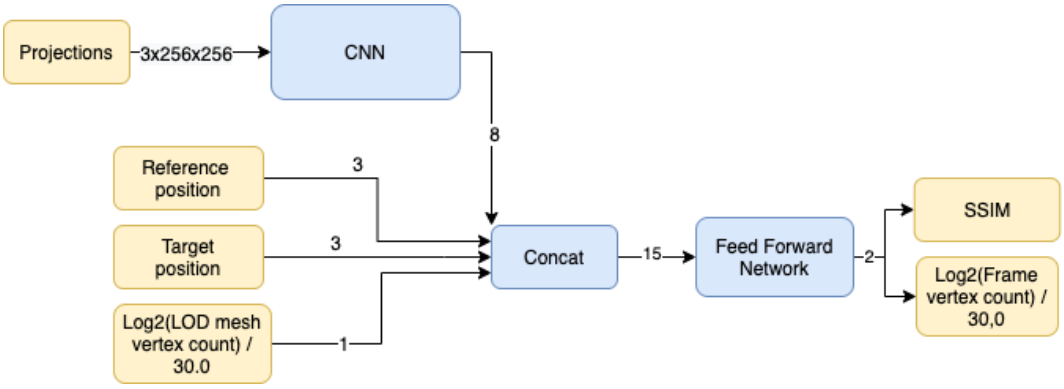

Figure 6.8: Architecture of our deep learning model

6.6.1 CNN feature extractor

The projections are fed into the Convolutional Neural Network whose task is to extract some features from the projections of the 3D mesh. The CNN's

architecture is described more in detail in figure 6.9: it consists of four blocks; each block comprises of a convolution layer, a ReLU activation function and a max pooling layer. The output of the CNN is a feature vector of length 8 which is the same number of 3D objects in our dataset. In future work, we may want to increase the number of 3D models in our training set as well as the CNN's output feature size for improving generalisation to different 3D objects.

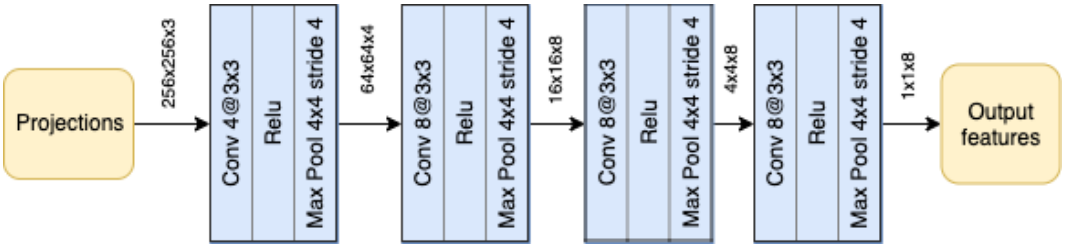

Figure 6.9: Diagram of the Convolutional Neural Network

6.6.2 Feed forward network

The extracted mesh features are subsequently concatenated with the reference and target positions, and the normalised logarithm of the total mesh vertex count of the target LOD. The result of this concatenation is a feature vector of length 15 forming the input for the Feed Forward Network whose architecture is illustrated in figure 6.10.

The Feed Forward Network is composed of four blocks, where each block consists of three layers: namely, a linear layer (also known as "fully connected" layer), a ReLU activation function, and a dropout layer. Each inner fully connected layer has 256 neurons whereas the output layer has 2 output neurons corresponding to the SSIM score and the normalised logarithm of the frame

vertex count.

The dropout layer randomly drops units (along with their connections) from the neural network during training. This prevents the units from co-adapting too much. This significantly reduces overfitting and gives major improvements over other regularisation methods. [22]

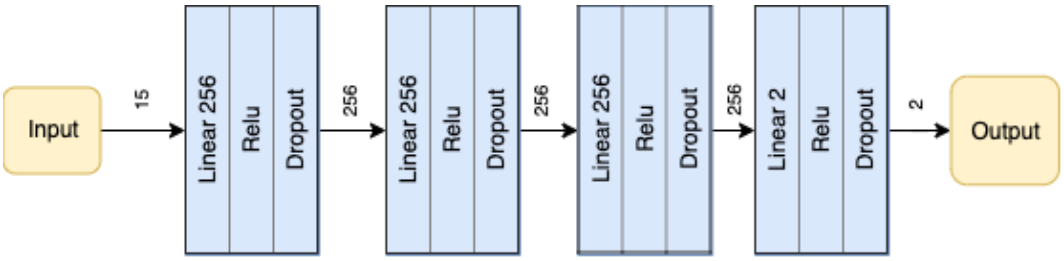

Figure 6.10: Diagram of the Feed Forward Network

Chapter 7

Experiments

In the following experiments, we illustrate some training, validation, and test results as well as the hyperparameters adopted in the experiments. We implemented and trained our deep learning model using the *PyTorch* framework, and developing in the Python programming language.

Furthermore, we developed a demo in Unity using our deep learning quality predictor to produce an LOD selection framework. Finally, we will show some analysis of our subjective experiments.

7.1 Training

In order to train our neural network model we used *Adam* optimisation algorithm. Adam optimiser is a stochastic gradient descent algorithm based on adaptive estimates of lower-order moments. It is straightforward to implement, is computationally efficient, has little memory requirements, and is well suited for problems that are large in terms of data and/or parameters. [21]

We have used a reduce-on-plateau learning rate scheduler. It reduces learning rate when a metric has stopped improving. This scheduler reads a metric

value and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced. [7]

In addition to the architecture, other parameters are fixed during the training phase: namely, the batch size, through which the gradient descent is applied, the initial learning rate, the multiplicative factor for the reduction, and the number of epochs. The values of these parameters are listed in table 7.1.

Table 7.1: Hyper-parameters

Parameter	**Value**
Batch size	4
Initial learning rate	0.001
Learning rate decay factor	0.7
Learning rate patience	10
Epochs	150

7.2 Results

7.2.1 Training and validation

In figure 7.1 we can notice a good training curve that converges to a low mean squared error. The validation curve reflects the training curve: the validation errors are not neither significantly larger nor smaller than the training errors. This indicates an absence of overfitting or underfitting.

Training and validation loss at the end of training are listed in table 7.2.

In table 7.3 we can see an example of prediction of a single minibatch of the validation set at the end of the training.

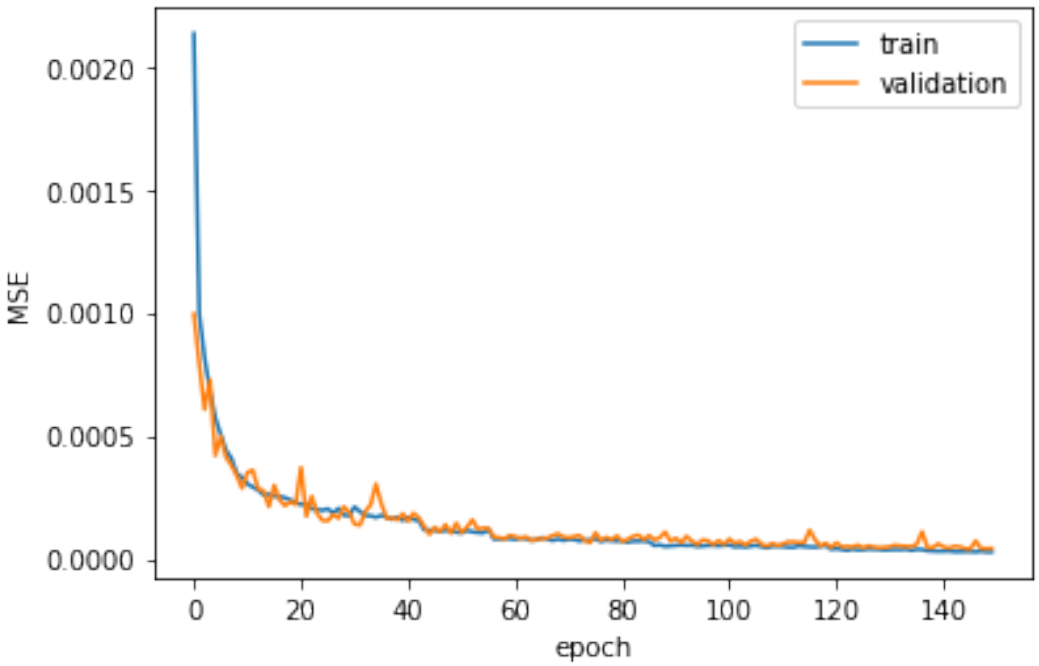

Figure 7.1: Training and validation history

7.2.2 Test

The following test is executed on a 3D model which is not present in the training set. We report in table 7.4 the mean squared error of our model's test performance and in table 7.5 an example of prediction of a single minibatch sampled from our test set. It is normal to see a larger error compared to the previous validation error, since our test set is generated from a 3D mesh our neural network has never been trained on. We can see that our network is able to retain some generalisation capability to different 3D meshes. This is a positive result if we consider that our training set is currently generated from only 8 different 3D models. In future work, we expect to improve the results by adding more 3D models in our training set.

Table 7.2: Final training and validation loss (150 epochs)

Loss	**MSE**
Training loss	2.8e-5
Validation loss	4.2e-5

Table 7.3: Example of prediction of a mini-batch (4 samples) in validation set

Predicted		**Ground truth**	
ssim	normalised vertex count	ssim	normalised vertex count
0.5407	0.6635	0.5305	0.6688
0.9057	0.6599	0.8985	0.6561
0.9589	0.6529	0.9646	0.6527
1.0103	0.8558	1.0000	0.8549

7.3 Demo

The previous results have shown that it is possible to predict quite accurately SSIM scores and the vertex count in a specific instant of an interactive 3D visualisation. In the following step, we decided to make a demo to see the deep learning model in action as an LOD selection framework in a 3D interactive application.

7.3.1 Exporting the deep learning model

After training we exported the final deep learning model in ONNX format. We chose this format because it can be easily imported into Unity using the *Barracuda* library.

The Barracuda package is a lightweight cross-platform neural network inference library for Unity. Many platforms are supported, including mixed

Table 7.4: Final loss on test set

Loss	**MSE**
Test loss	5.69e-4

Table 7.5: Example of prediction of a mini-batch (4 samples) in test set

Predicted		**Ground truth**	
ssim	normalised vertex count	ssim	normalised vertex count
0.8715	0.6653	0.7837	0.6649
0.9630	0.6510	0.9422	0.6562
0.9876	0.6161	0.9395	0.5198
0.9816	0.6163	0.9508	0.5198

reality headsets running Android operating system. Barracuda can run neural networks on both the GPU and CPU. [5]

7.3.2 Quality prediction

Objective During the interactive session, our objective is to pick the LOD that best optimise the successive frames. Every 1s, we run the neural network to compute a score for each LOD in order to determine the LOD with the highest score, which will be selected for rendering.

Input Once we imported into Unity the deep learning model and the 3D model we want to test on, we computed the CNN features by feeding the CNN the orthographic projections. Fortunately, this computation is executed only once, at the start of the interactive session, because it is independent from the camera position.

In addition to feeding our feed-forward neural network with the current

user position (called reference position), we also compute the target position by forecasting the user's position 0.1s into the future by taking into account the current camera speed. The pair of positions gives our deep learning model information on the current movement of the user, which will be used to compute how quickly the frames being rendered are changing in visual appearance, and in the end predict how much perceived lag a specific LOD would produce.

Lastly, the input to our neural network includes the normalised logarithm of the total vertex count of a specific LOD.

These information, namely, the CNN features, the pair of positions, and the mesh vertex count, are finally concatenated to form the input of our feed-forward neural network.

Output The output produced by the neural network is composed of the predicted SSIM score and the predicted normalised frame vertex count (FVC). This operation is repeated for each LOD of our 3D model and the computed indices are then used to assign an estimated QoE score to each LOD.

QoE score The QoE score is a heuristic function of the SSIM score and the FVC. If we take into consideration that each person may have different preferences regarding what is a good balance between lag and level of detail, we can see that it is hard to define a fixed function that works well for anyone. Therefore, we decided to designate the QoE score as a parametric linear function of the two indices, and the parameters can be tuned to fit the user's preferences.

$$\text{QoE score} = \alpha * \text{SSIM} - \beta * \text{FVC} \tag{7.1}$$

The SSIM index provides information on the visual quality of the 3D model from the current user viewpoint, whereas the FVC index gives information on

the frame's complexity. Higher SSIMs are better, whereas a higher FVC is worse because it would translate in lower Fps, hence the negative sign.

For this demo, after trying different values to see what seemingly works best, we settled on the parameter values: $\alpha = 3$ and $\beta = 1$.

Demo results After testing our demo, we noticed that during the interactive session our framework typically selects: a lower LOD when viewing the object from a distant position or when moving quickly around the object; a higher LOD when looking the 3D object from a close perspective or when the camera is stationary. These observations are what we expected given the previous considerations. The overhead that derives from running the deep learning model is reasonably limited thanks to the small and lightweight size of the neural network, the re-usability of the CNN features, and the amortisation of the costs by running the prediction every second instead of every frame.

7.4 Subjective experiments

In a thesis work completed in collaboration with Luca Hartz, also a Unipd near-graduate in Computer Engineering, we performed some experiments collecting subjective evaluations of multiple 3D models at different LODs from different people.

As we have previously mentioned, these subjective data will be valuable for correlating them with our QoE score prediction, in other words, we want to verify if our predictor can, in some way, predict the perceived QoE of any LOD.

We invited 52 different people to perform 15 interactive sessions (5 3D models, each at 3 different LODs) on the Oculus Quest 2 VR headset. We asked the subjects to provide an evaluation on a scale from 1 to 5 of the QoE

at the end of each interactive session.

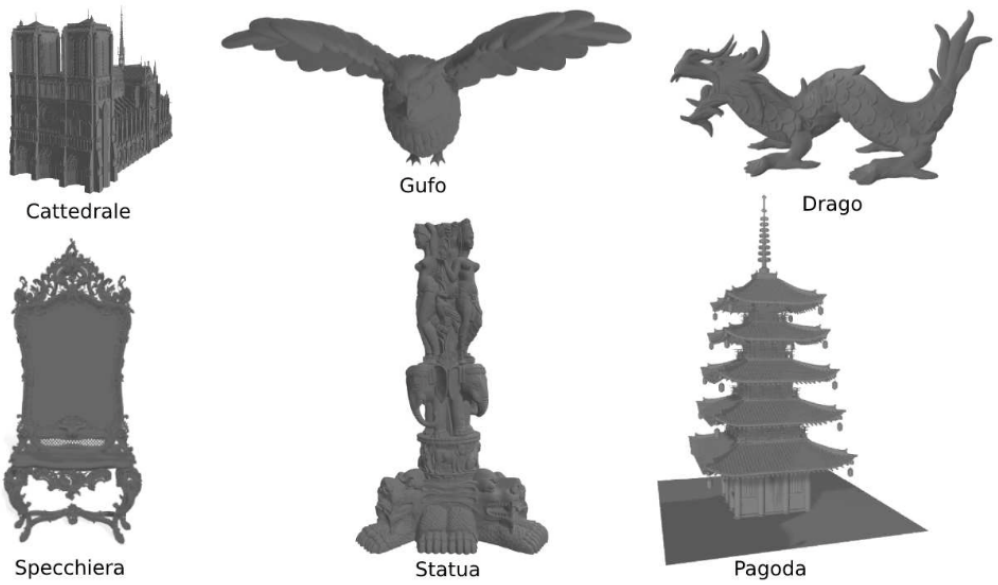

Figure 7.2: The 3D models we selected for the subjective experiments

Each interactive session lasted for about 15 seconds, followed by at least 3 seconds of interval between the visualisation and evaluation, for a total of about 5/6 minutes of interaction in the virtual environment. During the interactive sessions, while the participants were able to freely move around the object, we registered at each frame: the timestamp, the participant's position and orientation, and the frame rate.

Results After examining the collected subjective evaluations, we confirmed our initial hypothesis regarding the impact of frame rate on QoE, in particular, we observed a meaningful decrease in quality evaluation caused by a drop in frame rate. On the other hand, the level of detail of the selected 3D models does not affect the MOS score quite as significantly. [15]

As we have mentioned, from the 780 subjective ratings we have collected, we tried to correlate them with the scores predicted by our deep learning model. It is necessary to take into consideration that each subjective evaluation refers to a whole interactive session, whereas our QoE predictor gives a score at a given instant of the immersive experience. Therefore, we had to determine the average of QoE score prediction over all frames of an interactive session in order to compare it to the given subjective rating.

Unfortunately, we haven't noticed any considerable correlation between subjective and objective data. One of the reasons could be that the obtained subjective data could be insufficient or unreliable. Another hypothesis may be that the predictor is not capable of producing an accurate QoE score for the whole immersive session experience.

Chapter 8

Conclusions

The experiments' results on the neural network training proved that it is possible to accurately predict objective quality metrics such as SSIM and FPS. We have also seen that the predicting capabilities are not limited to just the 3D models used in the training set, but can also be generalised to different 3D objects thanks to the usage of orthographic triangle count projections.

As we have learned from previous studies, the visual quality and smoothness of animation are the most important factors to take into consideration when evaluating QoE. The importance of the listed factors are also supported by the fact that they can be controlled by designing an LOD selection framework. Therefore, we believe that the metrics predicted by our model are representative of the QoE in an interactive 3D graphics application.

In our demo application, the objective quality metrics predicted by our model have appeared to be effective in portraying the QoE. Our implementation has demonstrated that a deep learning assisted LOD selection framework can be useful in improving the QoE in an interactive 3D application. Moreover, the small overhead that comes from running the deep learning model does not affect noticeably the performance.

The impact of a low frame rate on the QoE is demonstrated by the analysis of the subjective experiments as we have observed a reduction in quality rating when there was a drop in FPS. However, it is still unclear to what extent the visual quality, ultimately measured by the SSIM index, affects the experience. Our attempt to correlate objective metrics with subjective ratings of VR experiences was inconclusive.

Therefore, some possible avenues for future research on the topic may include collecting more detailed subjective evaluations and carrying out a different subjective experiment modality. It would be interesting to test our deep learning assisted LOD selection framework against traditional frameworks such as distance-based LOD selection. Another interesting proposal would be to fine-tune our deep learning model using the collected subjective data.

In conclusion, the results of this work have shown that deep learning tools can be effective in optimising the quality of experience in a 3D interactive immersive application. We proposed an optimisation framework that looks promising and further research and experimentation are required to validate the results.

Bibliography

[1] 25 years later: A brief analysis of gpu processing efficiency. https://www.techspot.com/article/2008-gpu-efficiency-historical-analysis/.

[2] Adaptive bitrate streaming with hls and mpeg-dash. https://cloudinary.com/product_updates/plug_and_play_adaptive_bitrate_streaming_with_hls_and_mpeg_dash.

[3] Convolutional neural networks (cnns / convnets). https://cs231n.github.io/convolutional-networks/.

[4] Cross validation vs train validation test. https://stats.stackexchange.com/questions/410118/cross-validation-vs-train-validation-test.

[5] Introduction to barracuda. https://docs.unity3d.com/Packages/com.unity.barracuda@1.0/manual/index.html.

[6] Learning rate in machine learning. https://www.educative.io/edpresso/learning-rate-in-machine-learning.

[7] Pytorch. https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html.

[8] Unity. https://unity3d.com/beta/2020.1b.

[9] Unity at 10: For better—or worse—game development has never been easier. https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/.

[10] J. Leander A. C. Neves, I. Gonzalez. A new approach to damage detection in bridges using machine learning.

[11] Y. Bengio A. Courville, I. Goodfellow. *Deep Learning*. 2015.

[12] N Walker B Watson, V Spaulding. Evaluation of the effects of frame time variation on vr task performance.

[13] F. Chiariotti. Reinforcement learning algorithms for dash video streaming. Master's thesis, University of Padova.

[14] Alejandro Escontrela. Convolutional neural networks from the ground up. https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1, 2018.

[15] L. Hartz. Ottimizzazione della qualità soggettiva in dispositivi di realtà virtuale. Master's thesis, University of Padova.

[16] A. L. García J. G. Rodríguez. *Artificial Intelligence for Videogames with Deep Learning*. 2019.

[17] K Gorro J M Brotzer, E R Mosqueda. Predicting emotion in music through audio pattern analysis.

[18] Hung-Chi Lee James Davis, Yi-Hsuan Hsieh. Humans perceive flicker artifacts at 500 hz.

[19] J F Shepherd M Callahan, M J Cole. A meshing pipeline for biomedical computing.

[20] N. Locascio N. Buduma. *Fundamentals of Deep Learning : Designing Next-Generation Machine Intelligence Algorithms.* 2017.

[21] A Krizhevsky N Srivastava, G Hinton. Adam: a method for stochastic optimization.

[22] A Krizhevsky N Srivastava, G Hinton. Dropout: A simple way to prevent neural networks from overfitting.

[23] B. A. Watson R. Huebner, M. Reddy. *Level of Detail for 3D Graphics.* 2002.

[24] William E. Lorensen William J. Schroeder, Jonathan A. Zarge. Decimation of triangle meshes.

[25] E. P. Simoncelli Z. Wang, A. C. Bovik. Image quality assessment: From error visibility to structural similarity.