Deep 3D Model Optimization for Immersive and Interactive Applications

Elena Camuffo, Federica Battisti, Francesco Pham, Simone Milani
Department of Information Engineering, University of Padova
{elena.camuffo, federica.battisti, simone.milani}@unipd.it, francesco.pham@studenti.unipd.it
https://github.com/elenacamuffo/Deep-3D-optimization

***Abstract*—The growing diffusion of immersive and interactive applications is posing new challenges in the multimedia processing chain. When dealing with AR and VR applications, the most relevant aspects to consider are the (1) quality of the visualized 3D objects and (2) the fluidity in the visualization in case the user is moving in the environment. In this framework, we propose a deep learning based approach that estimates the optimal model parameters to be used in relation to the viewer's movement and the model characteristics and quality. The performed tests show the effectiveness of the proposed approach.**

***Index Terms*—3D model, immersive media, interactive media, optimization, quality**

I. Introduction

The recent diffusion of Virtual and Augmented Reality (VR and AR) applications on mobile and wearable devices has posed new and challenging problems to designer and app developers. Indeed, providing an immersive experience to the user implies a smooth rendering of 3D models, as well as an accurate registration of the displayed view with respect to the object location and the viewer pose and orientation [1]. The viewer is considered as the observer's viewpoint: it corresponds either to the user or to the camera depending on the rendering device (head mounted display or mobile device respectively). Such requirements have significant implications on the computational effort of the devices [2], the adopted transmission bandwidth (when the 3D model is streamed) [3] and on the resulting quality perceived by the user [4]. As a matter of fact, 3D model simplification and formatting [5] significantly affect this task since it allows reducing the total amount of triangles or 3D points with negligible quality difference with respect to the original 3D model [6]. To this purpose, several shape and appearance simplification solutions have been adopted [7] to adapt the Level-of-Details (LODs) along time according to users' proximity and interaction. Most of the previous works focus on adapting the cognitive load [8], predicting users' action in order to optimize the training experience [9] or minimizing the amount of transmitted information [10].

This paper presents a self-adaptive strategy for AR/VR applications that chooses the most appropriate LOD for a given model according to user movements and the displayed object characteristics. This estimation is performed by means of a shallow neural regression network that processes the time series of previous viewing positions and orientations, as well as an integral image of the model characterizing the number of hidden surfaces from different viewpoints. The proposed model was optimized for an embedded implementation and integrated into a Unity3D app that can be deployed on several mobile AR/VR devices. Experimental results show that the proposed solution performs well for several sequences.

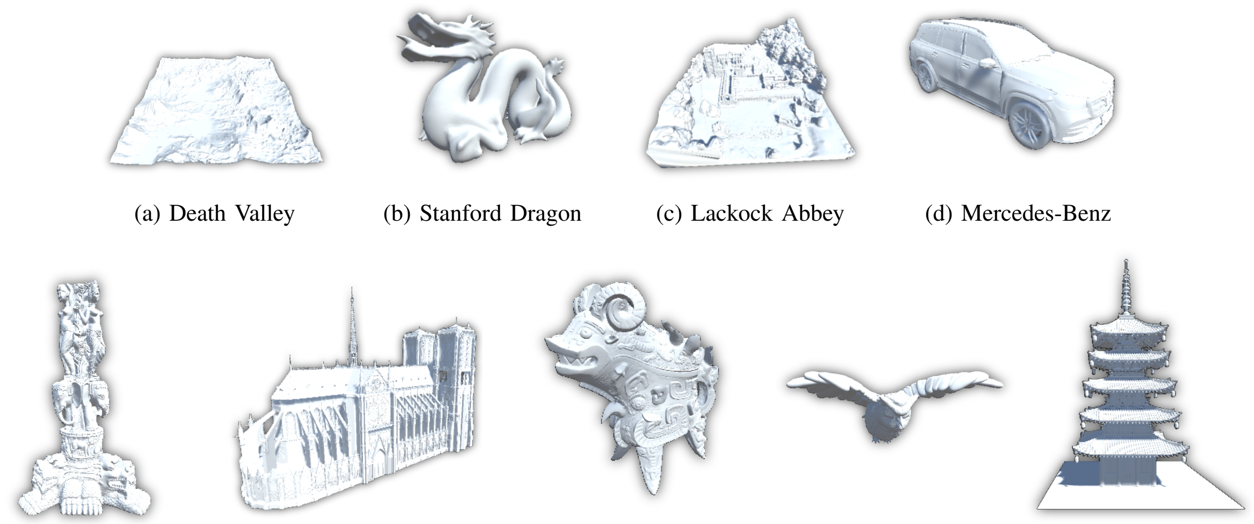
The remainder of the paper is organized as follows: Section II defines the problem of finding the optimal LOD as a rate shaping problem and overviews some of the related works proposed in literature. Section III overviews the preliminary steps required to build up the dataset, detailed in Section IV. Section V describes the adopted architecture, whose performance is explained in Section VI. Section VII draws the final conclusions.

II. Problem statement and related works

The LOD L of a 3D mesh model in a AR/VR applications strongly affects the system efficiency and the quality perceived by the end user. The adoption of a large number of triangles results to be unnecessary whenever the viewing point is distant since most of the details cannot be appreciated because of the small size of the rendered model [4]. On the contrary, a low LOD results in a poor visualization quality whenever the object is close to the viewer. As a matter of fact, the complexity of the visualized model must be accurately tuned and adapted according to the 3D model and user motion. This problem is very similar to the rate adaptation problem in video streaming [11]–[13]. In the AR/VR case, the perspective projection corresponds to the transmission channel (the more distant, the lower transmission capacity), the LOD (*i.e.,* the visual quality of the original displayed content) to the coding bit rate, the update frequency $\mathcal{V}$ of rendering (measured in Frames per Second (FPS)) to the frequency and length of stalls. For this reason, it is possible to model the quality-based optimization in a VR/AR application as a dual problem of quality maximization given complexity constraints on the LOD level:

$$\min L \quad \text{s.t.} \quad \begin{array}{l} Q(L, \mathbf{p}, \mathbf{o}) > Q_0 \\ \text{FPS}(L) > \text{FPS}_0 \end{array} \tag{1}$$

The quality function $Q(\cdot)$ is affected by the LOD L, the position $\mathbf{p}$ and the orientation $\mathbf{o}$ of the viewer with respect to the location of the virtual object, while the refresh rate FPS is related to the computational load on the GPU and affected



(a) Death Valley (b) Stanford Dragon (c) Lackock Abbey (d) Mercedes-Benz

(e) Thais Statue (f) Notre Dame (g) Lidded Ewer (h) Owl (i) Pagoda

Fig. 1: The set of 3D models that compose our dataset. Models from (a) to (g) compose the training set. Model (h) is used as validation set, and model (i) as test set. All the 3D models are considered with no attributes (*e.g.,* texture, normal mappings, etc.).

by the characteristics of the displayed 3D model. Q_0 and FPS_0 are the quality and frame rate at the original LOD L_0.

In literature, such problem has been investigated for videos as a quality maximization strategy under bandwidth constraints [14]. Several HTTP Adaptive Streaming (HAS) strategies propose packet dropping strategies to tune the transmission stream while minimizing the quality decrement [15]–[19]. Most of these strategies rely on linear programming solvers on simplified parametric rate and quality models. Recent works have addressed this problem by exploiting deep learning solutions [20]–[22].

This task has become even more challenging for AR/VR applications since 3D models are heterogeneous and require different rendering capabilities. In this contribution, we propose a deep learning approach that can predict the Structural Similarity Index (SSIM) score [23] of a model and the optimal number of vertices for 3D mesh representation from a specific viewpoint. Assuming that multiple version with varying LODs are available, our approach allows selecting the optimal versions of the model under different viewing conditions in immersive and interactive applications.

The following sections will provide further details about the overall procedure.

III. Scene analysis and evaluation metrics

The adopted experimental setup consists in a set of 9 standard three-dimensional mesh models (Fig. 1) obtained from the Stanford 3D Scanning Repository [24], and online 3D model platforms such as Sketchfab, Turbosquid, and CGtrader. The selected 3D models present different characteristics in terms of shape and complexity, to have a wide model variability within our dataset. For each model we generate three additional versions, using the *decimate modifier* in Blender (Fig. 3). Such generated models correspond to different LOD, L_i, of the original model, with $i = 1, 2, 3$, indicating the level of decimation. The original mesh corresponds to L_0 for $i = 0$.

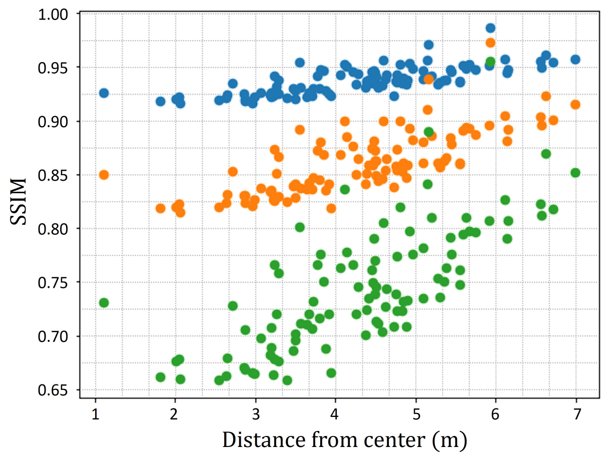
In order to optimize the model visualization, we considered the objective quality of the mesh through the computation of the SSIM, and the geometrical complexity of the model measured by the number of vertices used for each viewpoint as relevant elements. More details are given in the following sections.

A. Intra-view SSIM

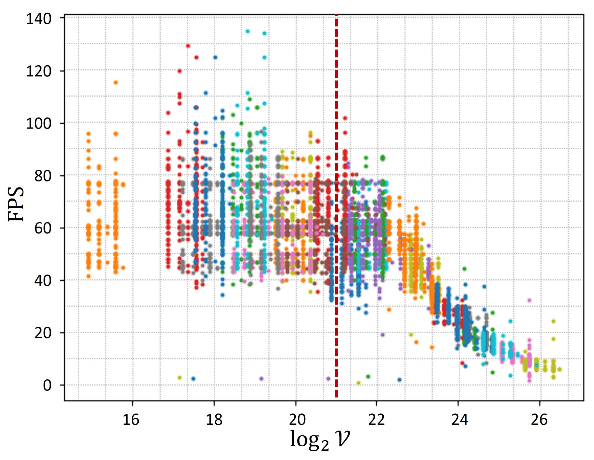
As preliminary study, we evaluate the objective quality of the 3D model through the SSIM [23]. The structural similarity is computed between a target and a reference view. Such views are rendered in Unity3D, using the original high-quality mesh at LOD L_0 for the reference view, and its lower resolution versions at LODs L_i with $i = 1, 2, 3$ for the target ones.

The viewpoints are generated for a set of random camera positions $\mathbf{p} = \{\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_n}\}$ around the 3D object. Such positions are uniformly distributed in the area around the 3D model, which is scaled to fit a $1\mathrm{m}\times1\mathrm{m}\times1\mathrm{m}$ box in the Unity3D reference system, and centered around the origin of the 3D axes, *i.e.,* positioned in $(0, 0, 0)$. This allows a uniform sampling of the quality and complexity of the 3D model viewed from any direction. Each camera pose includes both position and rotation coordinates (9 Degrees of Freedom) and generates a 256×256 pixel image. Rotation coordinates are not taken into account in the randomization since we assume that the viewpoint is always directed towards the 3D object.

The performed analysis shows that larger SSIM values are obtained when the camera is positioned farther away from the



(a) Intra-view SSIM vs distance



(b) FPS vs vertex count

Fig. 2: Correlation plots of different output measures: (a) Correlation between the Intra-view SSIM and the distance from the axes' center (m). The highest LODs L_1 (blue) achieves the highest values of SSIM, and the SSIM decreases while progressively lowering the LOD with L_2 (orange) and L_3 (green); (b) FPS-frame vertex count correlation. FPS and $\log_2(\mathcal{V})$ are inversely correlated, but FPS is upper-bounded to $\sim$ 60Hz for lower levels of $\log_2(\mathcal{V})$ ($\sim 2^{21}$). Different colors correspond to different 3D models: for each model 3 clusters (corresponding to the 3 LODs) are visible in correspondence of 3 different $\log_2(\mathcal{V})$; each cluster is composed of samples with the same LOD but different values of $\mathbf{p}_0$ and $\mathbf{p}_t$.

3D object while smaller SSIM values are obtained for less detailed objects (*i.e.,* with lower LOD). This can be noticed in Fig. 2a, where the blue dots refer to the SSIM values computed for L_1, the orange dots for L_2, and the green ones for L_3.

B. *Inter-view SSIM*

In a realistic scenario the user is usually free to move in the environment. This movement is simulated through a random walk, in a time interval spanning from T_0 to $T_0 + \delta T$, which provides uniformly distributed positions taking x, z values in the interval $[-1, 1]$ and $y \in [0, 1]$ (around the object).

To this aim, pairs of reference and target position $(\mathbf{p}_0, \mathbf{p}_t)$ are sampled among the generated positions. The reference and target camera positions are determined supposing that the distance of the target from the reference position is not larger than 0.3m in the Unity3D measurement system, *i.e.,* $||\mathbf{p}_t - \mathbf{p}_0|| \leq 0.3$. The selection of this maximum distance value is based on the assumption that in interactive applications, it is unlikely to have large and sudden camera jumps. Also in this case, we suppose that the user is always oriented towards the origin of the axes, *i.e.,* where the 3D model is placed. The faster the user is moving, the farther apart are the reference and target camera positions, and the lower is the SSIM. The computation is repeated varying the LOD in the target position, *i.e.,* the target frame is rendered from $\mathbf{p}_0$ with LOD L_0, while the target frame is rendered from $\mathbf{p}_t$ with LOD L_τ, where L_τ is sampled from $\{L_i\}_{i=1,2,3}$.

C. *Frame Vertex Count*

The user's perceived quality is also affected by the number of Frames per Second. However, such measure is strongly dependent on the hardware capabilities of the rendering device. Consequently, FPS is a variable measurement that can hardly be predicted in a platform-independent way: for this reason, in our experiments we considered instead the vertex count measure, which is strongly correlated to the number of FPS.

First, we consider for each model the *total vertex count* V, *i.e.,* the total number of vertices of the considered mesh. Then, we consider the *frame vertex count* as the number of vertices of the mesh that are visible from a specific rendering viewpoint and that directly affects the computational load of the rendering process. It is defined as $\mathcal{V}_i \subset V \mid \mathbf{p_i}$, where $\mathbf{p_i}$ is the rendering viewpoint. The larger is the number of vertices to be rendered, the longer it takes to render a single frame (and the lower is the refresh rate).

Fig. 2b justifies the selection of the *frame vertex count* as a platform-independent variable with respect to the FPS. The figure clearly shows that the correlation between the FPS and the base-2 logarithm of the *frame vertex count* $\mathcal{V}_i$. Below a certain threshold of vertex count (around $\sim 2^{21}$), the FPS is limited to $\sim$ 60fps, as a standard display refresh rate is 60Hz, and the FPS is capped to that frequency.

In addition, attributes like texture mapping and normals can considerably increase the computational load of rendering. Such measurements will be included in future research activities: in the current setup, objects are textureless and we are focusing only on the geometrical complexity.

D. *Orthographic Triangle Count Projections*

In order to complete the characterization of model complexity and rendering quality, some information about the

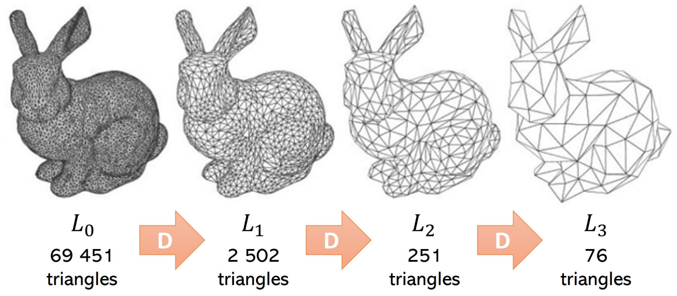


Fig. 3: Decimation procedure determines 4 different Levels of Detail for each 3D model. L_0 represents the highest LOD (original 3D model), L_3 the lowest LOD.

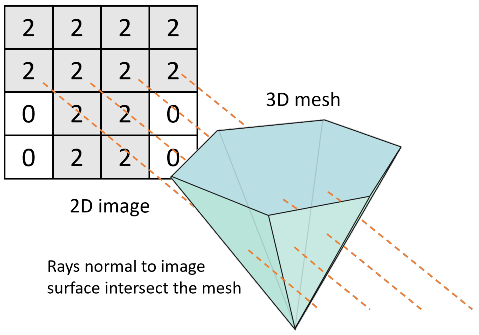


Fig. 4: Visual explanation of the OTC-projection mechanism.

three-dimensional structure of the 3D models needs to be considered.

We define Orthographic Triangle Count projection (OTC-projection) the procedure we used to extract information on the objects' 3D structure. It consists in extracting three 256×256 pixel bi-dimensional images from the three-dimensional view of the model, which is bounded into a 1m$\times$1m$\times$1m box and centered in $(0, 0, 0)$. Such images correspond to the three orthographic projections $\mathbf{I}_x$ $\mathbf{I}_y$ and $\mathbf{I}_z$ of the 3D model, along the x, y, z axes respectively. The value of each pixel in these projections is determined according to the number of triangular faces of the mesh that are intersected by the normal to the pixel surface. In other words, each pixel of each view is computed by casting an orthogonal ray passing through its center, and counting the number of triangles intersected by that ray (Fig. 4).

IV. Dataset Characterization

The dataset has been generated starting from the original 9 models described in Section III. In particular, 7 models have been used for training, 1 model (*Owl*) for validation and 1 model (*Pagoda*) for testing, obtaining respectively 8.6k, 1.4k and 1k samples. This test set is used to evaluate the deep learning model effectiveness for the generalization with different and new 3D models. Each dataset sample includes:

- **Projection images** $\mathbf{I}_x$, $\mathbf{I}_y$, $\mathbf{I}_z$: three 256×256 pixels bi-dimensional images corresponding to the OTC-projections of the 3D model;

TABLE I: Additional specifications on the network.

CNN parameters	7664 (6.01MB)
FFN parameters	76546 (0.30MB)
Training samples	8600
Validation samples	1400
Test samples	1000

TABLE II: Hyper-parameters of the network.

Parameter	**Value**
Batch size	256
Initial learning rate	0.001
lr decay (every 10 epochs)	0.99
Reduce-on-Plateau factor	0.7
Learning rate patience	10
Total epochs	150

- **Reference position** $\mathbf{p_0} = (x_0, y_0, z_0)$: the reference position of the camera in Cartesian coordinates;
- **Target position** $\mathbf{p_t} = (x_t, y_t, z_t)$: the target position of the camera in Cartesian coordinates;
- **Total vertex count** V: the total number of vertices in the 3D mesh for LOD L_τ (used at $\mathbf{p_t}$).

The other metrics are used as ground truth labels:

- **Inter-view SSIM**: the SSIM between the frame rendered from the camera in the reference position $\mathbf{p_0}$, at the reference LOD L_0, and the frame rendered from the camera in the target position $\mathbf{p_t}$ at the target LOD L_τ.
- **Frame vertex count** $\mathcal{V}_t$: the number of visible vertices among the total number of vertices of the mesh V at a given LOD L_τ, from a given rendering viewpoint $\mathbf{p_t}$.

V. Metrics Estimation Pipeline

The overall pipeline developed within our approach is reported in Fig. 5. It is composed of a Convolutional Neural Network (CNN) and a Feed Forward Network (FFN). The CNN branch encodes a set of 32 compact features in order to characterize model complexity starting from the OTC-projection images, mapped into 8 bit depth intensity maps.

CNN architecture: 4 convolutional layers are stacked in a typical encoder structure. The convolutional layers have padding 1 and respectively 4, 16, 16, 32 channels; each is followed by a ReLU activation function and a 4x4 Max Pooling layer with stride 4. The 32-sized feature vector produced at the output is concatenated with the reference position $\mathbf{p_0}$, the target position $\mathbf{p_t}$, and the total vertex parameter $V_{in} = (1/V_{MAX}) \log_2(V)$, which corresponds to the base-2 logarithm of the total vertex count V at target LOD L_τ normalized over $V_{MAX} = log_2(2^{30}) = 30$, *i.e.,* the nominal maximum exponent of the total number of vertices (given the statistics on our set of models).

FFN architecture: The FFN is built up of 3 block units, with the first two blocks composed of a Multi-Layer-Perceptron (MLP) with ReLU activation function and dropout with rates 0.2 and 0.1, respectively. The last block, instead, consists in a MLP followed by a Sigmoid activation. The

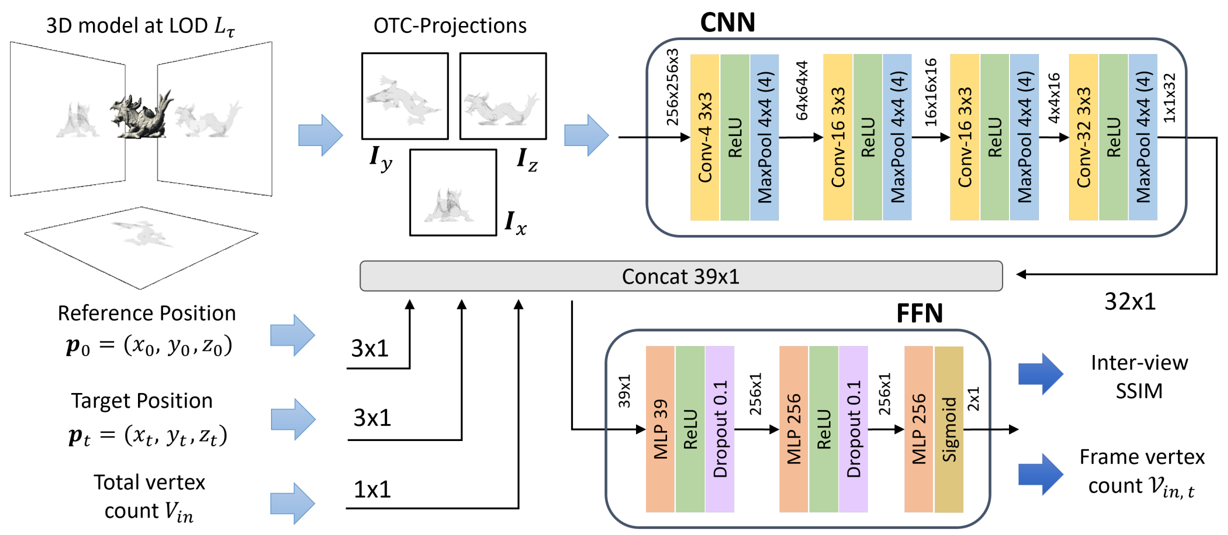


Fig. 5: Pipeline of the proposed approach. Three OTC-projections $\mathbf{I}_x, \mathbf{I}_y, \mathbf{I}_z$ are extracted from the 3D model and fed to a CNN encoder. The output of the CNN is concatenated with additional features, *i.e.,* the reference position $\mathbf{p}_0$, the target position $\mathbf{p}_t$ and the normalized logarithm of the total mesh vertex count at the target LOD L_τ, V_{in}. Such vector is fed to a FFN optimized on the SSIM score and on the normalized logarithm of the frame vertex count $\mathcal{V}_{in,t}$ metrics.

network produces jointly the prediction of the inter-view SSIM and a normalized frame vertex log-count, *i.e.,* $\mathcal{V}_{t,in} = (1/V_{MAX})\log_2(\mathcal{V}_t)$, where the target position is $\mathbf{p_t}$.

Training procedure: The whole network is trained minimizing MSE loss between estimated and ground truth parameters using Adam optimizer. The network has a total of 84210 parameters (all trainable) for a total occupation of 6.31MB (Table I). The network hyperparameters are reported in Table II. The learning rate is adapted with a reduce-on-plateau scheduler reducing its value whenever there is no loss reduction for *patience* number of epochs. All computations were implemented integrating Unity3D version 2020.1.11.f1 and PyTorch v1.11. The Training has been performed on a Tesla T4 GPU, and it takes almost 3.6 seconds to train a single epoch. Instead, to perform an evaluation, it takes just a few milliseconds.

VI. Experimental results

In this section we report the tests implemented to evaluate the performances of the proposed approach.

Table III and Table IV include the results obtained on the validation set and test set models, respectively. More specifically, we report the running average of the ground truth and predicted values for the Inter-view SSIM and the normalized *frame vertex count* $\mathcal{V}_{in}$ for each target LOD (with the running average of their respective estimation errors Δ).

Very satisfactory results are achieved on the *frame vertex count* $\mathcal{V}_i$ predictions, as we obtain an error of $2^{30\times\mathcal{V}_{in,test}} \approx 5.86$ vertices on the test set and of $2^{30\times\mathcal{V}_{in,val}} \approx 1.07$ vertices on the validation set. Considering that the minimum number of vertices $\mathcal{V}$ rendered from a specific viewpoint obtained in our dataset is larger than $2^{14} = 16384$, the prediction error found

TABLE III: SSIM and $\mathcal{V}_{in}$ results obtained on the validation set (*Owl*), averaged on the target LOD.

	SSIM			$\mathcal{V}_{in}$		
L_t	True	Pred.	$\Delta\downarrow$	True	Pred.	$\Delta\downarrow$
L_0	0.9498	0.9528	**0.0083**	0.8186	0.8198	**0.0030**
L_1	0.9514	0.9532	**0.0071**	0.7943	0.7955	**0.0028**
L_2	0.9506	0.9510	**0.0079**	0.7667	0.7682	**0.0026**
L_3	0.9528	0.9530	**0.0087**	0.7308	0.7325	**0.0028**
	0.8436	0.8532	**0.0187**	0.7212	0.7223	**0.0034**

TABLE IV: SSIM and $\mathcal{V}_{in}$ results obtained on the test set (*Pagoda*), averaged on the target LOD.

	SSIM			$\mathcal{V}_{in}$		
L_t	True	Pred.	$\Delta\downarrow$	True	Pred.	$\Delta\downarrow$
L_0	0.9001	0.8660	**0.0368**	0.7222	0.7270	**0.0056**
L_1	0.9029	0.8704	**0.0353**	0.6969	0.6894	**0.0076**
L_2	0.9000	0.8675	**0.0345**	0.6701	0.6602	**0.0099**
L_3	0.9004	0.8711	**0.0327**	0.6450	0.6340	**0.0110**
	0.9008	0.8687	**0.0349**	0.6850	0.6766	**0.0085**

is negligible. The prediction of the Inter-view SSIM proves to be effective, regardless a slight performance decrease for the test set.

The effectiveness of our approach is proved reporting predictions at different distances, we report the estimation accuracy whenever the user is far or close to the object in Table V and Table VI. We select two possible distance ranges for the reference and target positions with respect to the object, denoted as *far* (f) if $\sqrt{x_i^2 + z_i^2} > 0.5$ an *close* (c) if $\sqrt{x_i^2 + z_i^2} \leq 0.5$, with $i \in \{0, t\}$. The *far-to-far* case obtains the largest values of SSIM since the model complexity is not completely distinguishable when the camera is far from the object. On the contrary, the *close-to-close* case obtains lower

TABLE V: SSIM and $\mathcal{V}_{in}$ results obtained on the validation set (*Owl*), averaged in ranges of reference-target positions, classified as *far* (f) and *close* (c).

		SSIM			$\mathcal{V}_{in}$		
$\mathbf{p}_0$	$\mathbf{p}_t$	True	Pred	$\Delta\downarrow$	True	Pred	$\Delta\downarrow$
f	f	0.8667	0.8705	**0.0177**	0.7227	0.7236	**0.0051**
c	f	0.7312	0.7327	**0.0155**	0.7122	0.7125	**0.0045**
f	c	0.7466	0.7274	**0.0218**	0.7146	0.7224	**0.0078**
c	c	0.7583	0.7687	**0.0279**	0.7158	0.7174	**0.0043**

TABLE VI: SSIM and $\mathcal{V}_{in}$ results obtained on the test set (*Pagoda*), averaged in ranges of reference-target positions, classified as *far* (f) and *close* (c).

		SSIM			$\mathcal{V}_{in}$		
$\mathbf{p}_0$	$\mathbf{p}_t$	True	Pred.	$\Delta\downarrow$	True	Pred.	$\Delta\downarrow$
f	f	0.9124	0.8742	**0.0417**	0.7857	0.7073	**0.0217**
c	f	0.8476	0.7612	**0.0864**	0.6876	0.7046	**0.0170**
f	c	0.8569	0.7568	**0.1001**	0.6706	0.6977	**0.0271**
c	c	0.8583	0.8159	**0.0560**	0.6818	0.6995	**0.0176**

SSIM values. The overall performance follows the trend of the ground truth SSIM. Get worse when moving *far-to-close* since models' complexity makes the estimation hard.

In order to verify the proposed system in a real scenario, we implemented a demo using the *Barracuda* library in Unity3D. The main capability of the demo, is the automatic selection of the LOD for the 3D model, based on the output of the network. Through empirical tests, we observed that lower LODs are selected when the 3D object is distant from the user, or when the user is moving quickly around the object, and higher LODs when the 3D object is viewed from a close perspective, or when the user is still. Moreover, as mentioned in Section V, the time overhead that derives from running the deep learning model is reasonably limited thanks to the small and lightweight size of the neural network.

VII. Conclusions

In this paper we presented a deep learning based approach that aims at optimizing the visualization of 3D objects in an interactive scenario, adaptively selecting the most suitable set of parameters. The approach is based on the prediction of the SSIM and the *frame* vertex count $\mathcal{V}$ (strictly correlated with the FPS) metrics, in order to provide the best LOD for 3D meshes according to the user's movement. The obtained results show good performances, confirmed by the interactive demo application. In future work, we foresee to extend the number of adopted 3D models, as well as their complexity, by considering attributes such as texture and normal maps beside the geometrical features. Moreover, we foresee the implementation of subjective tests to collect the quality as perceived by the users.

References

[1] Sebastiano Verde, Marco Marcon, Simone Milani, and Stefano Tubaro, "Advanced assistive maintenance based on augmented reality and 5g networking," *Sensors*, vol. 20, no. 24, 2020.

[2] Fabio Capraro and Simone Milani, "Rendering-aware point cloud coding for mixed reality devices," in *2019 IEEE International Conference on Image Processing (ICIP)*, 2019, pp. 3706–3710.

[3] Ana De Abreu, Gene Cheung, Pascal Frossard, and Fernando Pereira, "Optimal lagrange multipliers for dependent rate allocation in video coding," *Signal Process. Image Commun.*, vol. 63, pp. 113–124, 2018.

[4] Alireza Javaheri, Catarina Brites, Fernando Pereira, and João Ascenso, "Point cloud rendering after coding: Impacts on subjective and objective quality," *IEEE Trans. Multim.*, vol. 23, pp. 4049–4064, 2021.

[5] Elena Camuffo, Daniele Mari, and Simone Milani, "Recent advancements in learning algorithms for point clouds: An updated overview," *Sensors*, vol. 22, no. 4, 2022.

[6] Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine, "Appearance-driven automatic 3D model simplification," in *Proc. of Eurographics Symposium on Rendering 2021*, Apr. 2021.

[7] Filip Biljecki, Hugo Ledoux, and Jantien Stoter, "Redefining the level of detail for 3D models," *GIM International*, vol. 28, pp. 21–23, 11 2014.

[8] Jayfus T. Doswell and Anna Skinner, "Augmenting human cognition with adaptive augmented reality," in *Foundations of Augmented Cognition. Advancing Human Performance and Decision-Making through Adaptive Systems*, 2014, pp. 104–113.

[9] Neil Vaughan, Bodgan Gabrys, and Venketesh N. Dubey, "An overview of self-adaptive technologies within virtual reality training," *Computer Science Review*, vol. 22, pp. 65–87, 2016.

[10] Hakran Kim, Yongik Yoon, and Hwajin Park, "Adaptation method for level of detail (LOD) of 3Dcontents," in *2007 IFIP International Conference on Network and Parallel Computing Workshops (NPC 2007)*, 2007, pp. 879–884.

[11] Truong Cong Thang, Hung T. Le, Anh T. Pham, and Yong Man Ro, "An evaluation of bitrate adaptation methods for HTTP live streaming," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 693–705, 2014.

[12] Simone Milani and Giancarlo Calvagno, "Distributed multiple description video transmission via noncooperative games with opportunistic players," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 1, pp. 125–138, 2015.

[13] Simone Milani and Giancarlo Calvagno, "A cognitive approach for effective coding and transmission of 3d video," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 7S, no. 1, nov 2011.

[14] Ozgur Oyman and Sarabjot Singh, "Quality of experience for HTTP adaptive streaming services," *IEEE Communications Magazine*, vol. 50, no. 4, pp. 20–27, 2012.

[15] Iraj Sodagar, "The MPEG-DASH standard for multimedia streaming over the Internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, 2011.

[16] Babak Taraghi, Minh Nguyen, Hadi Amirpour, and Christian Timmerer, "Intense: In-depth studies on stall events and quality switches and their impact on the quality of experience in http adaptive streaming," *IEEE Access*, vol. 9, pp. 118087–118098, 2021.

[17] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 187–198, aug 2014.

[18] Dang H Nguyen, Minh Nguyen, Nam Pham Ngoc, and Truong Cong Thang, "An adaptive method for low-delay 360 VR video streaming over HTTP/2," in *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*, 2018, pp. 261–266.

[19] Cong Wang, Divyashri Bhat, Amr Rizk, and Michael Zink, "Design and analysis of QoE-aware quality adaptation for DASH: A spectrum-based approach," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 13, no. 3s, jul 2017.

[20] Matteo Gadaleta, Federico Chiariotti, Michele Rossi, and Andrea Zanella, "D-DASH: A deep Q-learning framework for DASH video streaming," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 703–718, 2017.

[21] Jie Liu, Xiaoming Tao, and Jianhua Lu, "QoE-oriented rate adaptation for DASH with enhanced deep Q-learning," *IEEE Access*, vol. 7, pp. 8454–8469, 2019.

[22] Tho Nguyen Duc, Chanh Tran Minh, Tan Phan Xuan, and Eiji Kamioka, "Convolutional neural networks for continuous QoE prediction in video streaming services," *IEEE Access*, vol. 8, pp. 116268–116278, 2020.

[23] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[24] "The Stanford 3D Scanning Repository," http://http://graphics.stanford.edu/data/3Dscanrep/, Accessed: 2022-05-30.