Marco Rossoni
Department of Mechanical Engineering,
Politecnico di Milano,
Via La Masa 1,
Milano, Italy
email: marco.rossoni@polimi.it

Matteo Pozzi
Department of Electronics, Information and Bioengineering,
Politecnico di Milano,
Via Giuseppe Ponzio 34,
Milano, Italy
email: matteo13.pozzi@mail.polimi.it

Giorgio Colombo
Department of Mechanical Engineering,
Politecnico di Milano,
Via La Masa 1,
Milano, Italy
email: giorgio.colombo@polimi.it

Marco Gribaudo
Department of Electronics, Information and Bioengineering,
Politecnico di Milano,
Via Giuseppe Ponzio 34,
Milano, Italy
email: marco.gribaudo@polimi.it

Pietro Piazzolla[1]
Department of Mechanical Engineering,
Politecnico di Milano,
Via La Masa 1,
Milano, Italy
email: pietro.piazzolla@polimi.it

Physically-based Rendering of Animated Point Clouds for eXtended Reality

Point cloud 3D models are gaining increasing popularity due to the proliferation of scanning systems in various fields, including autonomous vehicles and robotics. When employed for rendering purposes, point clouds are typically depicted with their original colors acquired during the acquisition, often without taking into account the lighting conditions of the scene in which the model is situated. This can result in a lack of realism in numerous contexts, especially when dealing with animated point clouds used in eXtended Reality applications, where it is desirable for the model to respond to incoming light and seamlessly blend with the surrounding environment.

This paper proposes the application of Physically Based Rendering (PBR), a rendering technique widely used in Real-Time Computer Graphics applications, to animated point cloud models for reproducing specular reflections, and achieving a photo-realistic and physically accurate look under any lighting condition. To achieve this, we first explore the extension of commonly used animated point cloud formats to incorporate normal vectors and PBR parameters, like roughness and metalness. Additionally, the encoding of the animated environment maps necessary for the PBR technique is investigated. Then, an animated point cloud model is rendered with a shader implementing the proposed PBR method. Finally, we compare the outcomes of this PBR pipeline with traditional renderings of the same point cloud produced using commonly used shaders, taking into account different lighting conditions and environments. Through these comparisons, we demonstrate how the proposed PBR method enhances the visual integration of the point cloud with its surroundings. Furthermore, it will be shown that using this rendering technique it is possible to render different materials, by exploiting the features of PBR and the encoding of the surrounding environment.

Keywords: Point Cloud, Real-Time Rendering, eXtended Reality

1 Introduction

Rendering has been a key concept in Computer Graphics since its very first years. Many different techniques have been developed through the years, going from simple algorithms generating 8-bit "pixelated" colors to the most recent ones, showing realistic lighting effects. Physically Based Rendering (PBR) is a technique born in the '80s aiming to achieve photorealistic lighting and is currently widely used in many applications such as videogames, technical visualization, design and many other fields involving the creation of digital images.

Historically, Computer Graphics pipelines have been optimized to work with polygonal meshes, which are 3D models represented by a collection of vertices, edges and polygonal faces, in most cases triangles [1]. Meshes are still the most widely used type of 3D model in rendering, as hardware and accelerators have been highly optimized to work with such primitives in an efficient way. Thanks to the spreading of 3D scanning system, huge research effort has been directed to a different kind of 3D model based only on point primitives, known as *Point Clouds* (PC) [2]. These primitives have several advantages over polygonal meshes, such as the possibility to acquire them automatically and in a relatively short time, thus saving designers' work time. More importantly, since a point cloud has no connectivity information, points can be stored and transmitted in any order, as long as the whole set is preserved. This suggests that point clouds could be heavily adopted in Computer Graphics applications as a valid alternative to polygonal meshes.

To date, point clouds are usually visualized with simple rendering techniques able to display the colors of the models as they were at acquisition time only [1]. For some use cases, such as the point clouds acquired by autonomous cars where colors are used to distinguish obstacles on the path, there is no advantage in using advanced rendering techniques. However, there exist other use cases, especially those involving animations, where point clouds contextualization within a given environment is seek to obtain the higher photo-realism possible. This is the case with movies, immersive experiences in 360 degrees Virtual Reality (VR) such as virtual tours and games, or a mix between traditional 3D models and VR [3].

In this work, we focus on animated point clouds real-time photo-realistic rendering, with a focus on preserving the specular reflections saved at capture time. Usually, research about the rendering of points clouds is centered around static models, as noted by [4], so our efforts are directed toward improving the realism of animated and live captured scenes, implementing a real-time software rendering method based on Physically Based Rendering (PBR). Our method is able to incorporate environmental light contribution that may come from other static 3D meshes in the scene, 360 degrees background textures, or live video streams, thus providing useful

[1]Corresponding Author.
Version 1.18, September 29, 2023

support for eXtended Reality (XR) applications where a realistic integration of the model with environmental lighting is desirable.

The paper is organized as follows. Section 2 presents the main concepts and a brief state of the art. Section 3 describes the proposed method as well as all the steps to render an animated point cloud using a dedicated PBR shader. Section 4 includes a comparative study between the PBR renderings and the traditional pipeline considering different lighting conditions and environments. Finally, Section 5 contains some final considerations about the obtained results and discusses the limitations of this study, leaving space for future developments.

2 Related Works

Physically Based Rendering is a widely used collection of rendering techniques that mimics how light interacts with surfaces in a more physically realistic way compared to other shading models [5]. In [6], the authors investigated how those techniques have been applied to the real-time visualization of polygonal meshes in video games and movie production. The main research efforts in the domain of point cloud are directed to the acquisition of the point clouds [7], methods for generating a connected mesh from point cloud[8], point cloud compression methods [9,10] and rendering techniques [4].

When it comes to applications, most of the time point clouds are usually rendered to look exactly as if they were under the same lighting conditions as during the acquisition. Despite this is not a concern when the application does not require enhanced realism, it is not always the case when realism is a "must-have" feature. For example, Virtual, Augmented and Extended Reality may require static or animated models to be rendered as realistically as possible under dynamic lighting conditions [11]. The first attempt to use realistic rendering techniques for point cloud visualization is detailed in [12]. Further advances, including using PBR along with Image Based Lightning (IBL) [13], have already been made. One of the most notable studies has been carried out by Sun et. al. [14] which proposed a scanning technology, called LightStage, able to scan human faces when lit from any possible lighting direction and render them faithfully in different realistic virtual conditions. Exploiting a similar idea, this work extends the approach proposed by [14] to render animated point clouds under any lighting conditions.

Point clouds are a set of high-density individual points used to represent volumetric visual data, which can be computer-generated or directly captured from the real world. All these points carry various attributes about the properties of the object they are representing, like the basic position in the space (x, y, z coordinates) and eventually its color, surface normal, etc. [15]. They can be an alternative to polygonal meshes when representing 3D models, with the advantage that they can be directly sampled from the real world with cameras, and processed in real-time, without the need of reconstructing the surface. Due to their integration with graphics pipelines and surface representations, polygonal meshes are still widely used in all computer-graphics applications. However, point clouds are getting more and more popular in virtual and mixed reality applications thanks to their flexibility [16]. Since point clouds provide immediate depth information also, they find use in autonomous vehicle applications to detect objects, navigation, and localization[17]. In the construction industry, point clouds are used to reconstruct 3D models of buildings used for geometry quality inspection tasks and as a support in maintaining cultural heritage buildings[18]. Point cloud data can be acquired in different ways: using *passive methods*, which consists of multiple cameras placed around the subject while depth is computed using various methods ranging from image matching to spatial triangulation like in [19], [20] and [21], or using *Active methods*, that leverages on sensors able to collect depth data, like Intel Realsense[22], Microsoft Kinect [23] among others. To create high-quality point clouds, industries are recently investing in volumetric studios, where both methods can be used, also in combination, like the 8i Studios[24].

When considering animated point clouds, the computational effort can be quite bulky, especially for real-time rendering applications. Fortunately, model data tend to redundancy across the frames, and thus optimization techniques can be adopted. Point Cloud Compression, hence, became an interesting topic in the last years, and standardization for Point Cloud Compression (PCC) was completed by the Moving Picture Experts Group (MPEG)[2] in 2020[25]. The standard offers two different approaches: one for sets with a uniform distribution of points (V-PCC) and another for sets with a sparse distribution (G-PCC). The idea behind V-PCC is about converting a point cloud from 3D to 2D, by dividing it into connected regions named 3D patches (clusters) and projecting each of them independently into a 2D patch with orthogonal projections. Each patch can then be packed into images, compressed and encoded with any existing or future video codecs[26]. However, while the subsequent decoding result is generally bit-exact, the reconstruction can lead to a slightly different geometry with respect to the original one, introducing artifacts due to quantization errors. Studies to remove such artifacts are ongoing. For example, in [27] a solution using deep learning based on a U-Net architecture is presented. On the contrary, the G-PCC solution converts a 3D point cloud into a 2D representation and a geometry-based encoding serializes the 3D model by using an octree data structure, leveraging on a technique conceptually similar to voxelization[28]. In sparse PCs, only 1% of voxels are occupied, making the octree representation very convenient. Since more points can be mapped to the same sub-cube, the number of points for each sub-cube is also arithmetically encoded [9]. For a more detailed overview on the G-PCC standard the reader can refer to [29]. Although G-PCC can achieve remarkable performances, it might still lead to serious artifact issues during the attribute compression task, especially for low-bitrate scenarios. A first study has been made by Sheng et. al. [30] where a solution using a Multi-Scale Graph Attention Network is used to remove artifacts on compressed attributes. In [31], instead, an end-to-end framework based on a deep neural network achieved efficient compression results. The study showed that this method outperforms the G-PCC standard with a good margin of at least 60% BD-Rate. As eXtended Reality (XR) applications require uniform and dense PCs to be rendered at a satisfactory frame rate [4], V-PCC will be adopted in this work.

Starting from the idea presented in [14], this work extends the current approaches to render point clouds, making them more suitable for XR applications. Specifically, the current approaches for rendering consider static point clouds and rasterization-based methods, not able to make the point cloud integrated with the surrounding virtual environment. The approach proposed in this paper goes beyond these limitations, by considering *animated* point clouds surrounded by a given environment rendered with a *PBR, real-time* rendering engine.

3 Methods and Tools

This section provides a detailed description of the proposed methodology used to generate and visualize animated point clouds rendered with PBR. The literature about the rendering of points clouds is centered around static models [4], considering the color and lighting condition at the acquisition time only. In other words, it means that only three out of the seven channels required by PBR are actually used (see Sec. 3.2). The approach discussed in the following section, not only considers animated point clouds but allows them to be embedded in a virtual environment and react to the different lighting conditions in which they are immersed in. To do so, all the seven channels required by the PBR approach have been integrated with the mpg-vpcc compression method and sent over to the rendering engine. As outlined, the scene represented with the animated point cloud can be either produced with 3D animation or captured from a real-life environment, containing any moving agent. Hence, despite the paper's aims at determining whether this

[2] https://mpeg-pcc.org/

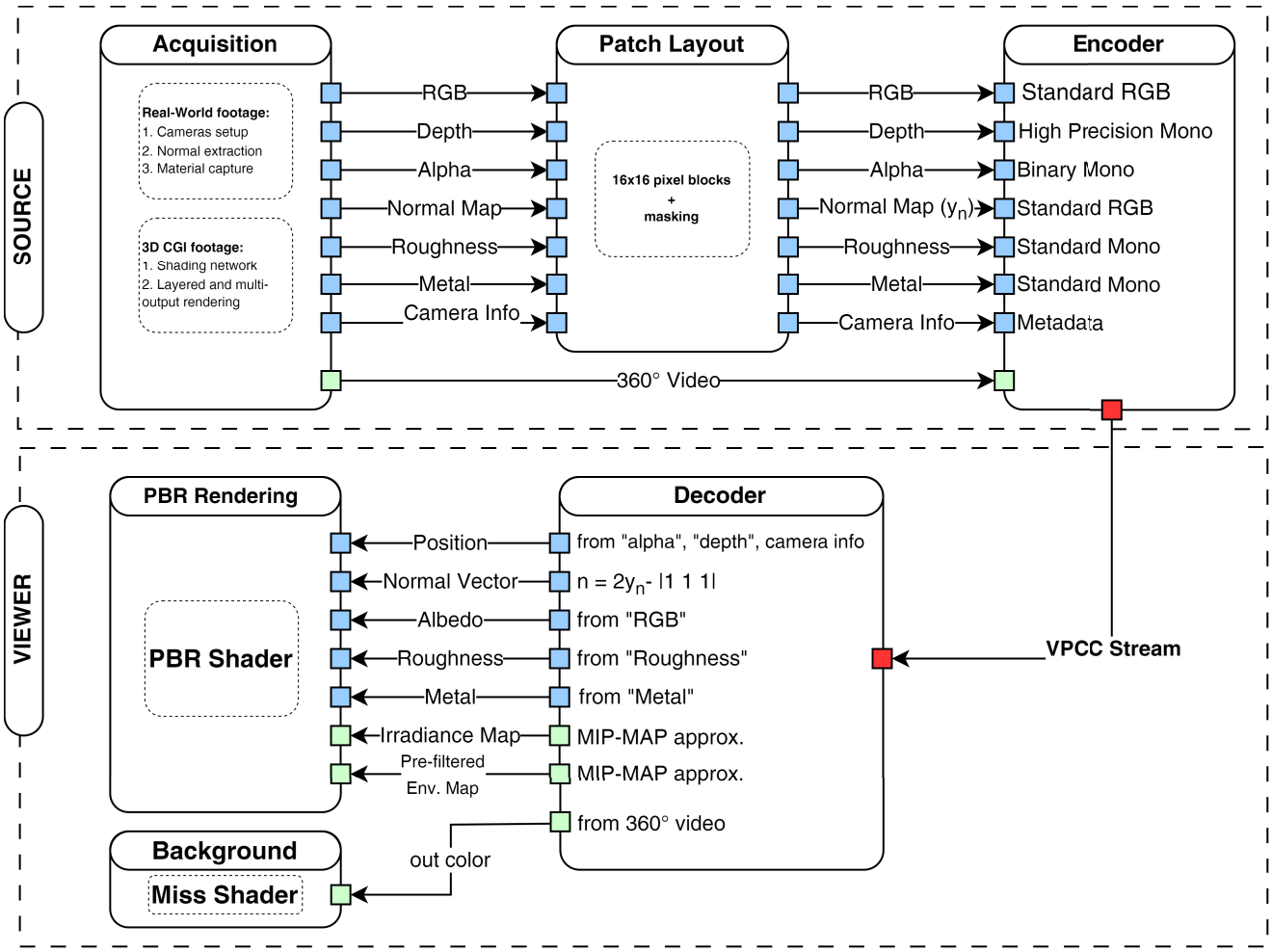


Fig. 1 The architecture of the proposed method. The source is referring to the point cloud data, the viewer is the real-time rendering engine. Arrows starting and ending in the green boxes represent the *backgroud* while the blue ones represent the *focus object* information flows.

approach can be a viable solution for real-time rendering, a brief discussion on how the point clouds can be generated in the two scenarios will be also included.

Figure 1 shows the complete architecture. In particular, there are two sets of information to be taken into account when rendering a scene: the *focus object* and the *background* whose information flows are highlighted by the blue and green boxes respectively. The former is the animated object of the scene: the viewer can move around and experience it from arbitrary directions by navigating the 3D environment. The latter represents the environment surrounding the PC under consideration, determining the background of the scene and the color of the pixels not covered by the focus object.. The PBR algorithm considered in this work will require:

(1) the *position* of each point of the cloud as a three-component vector that allows to place it properly in world space;
(2) the *diffuse* color as a three-component RGB value that defines the main color of the object in the considered point. The diffuse color is also known as *albedo*;
(3) the *normal vector direction* that represents the surface direction for each point. This is used by the PBR algorithm to compute the main features of the final color;
(4) the *metalness* of surface which defines how metallic the surface of the material is. It can be defined either as a floating-point value between 0 and 1 or directly sampled from a texture to render more complex objects with points of varying metalness along the surface.
(5) the *roughness* defining how smooth the surface is. As for metalness, it can be defined as a floating-point value ranging from 0 to 1 or be sampled from a texture map. In the latter case, the roughness values are usually sampled from the same map used for the metalness parameter. Depending on the rendering engine, the *smoothness*, which is the inverse of the *roughness*, could be exposed.
(6) the *irradiance map* and the *pre-filtered environment maps* to consider the specular reflections and perform image-based lighting of the point [32], a technique to illuminate scenes and objects grounding on images of light taken from the real world.

It is worth noticing that conventional point cloud rendering techniques exploit only the first two components of the previous list. The procedure also assumes that all these data are available for each point in each frame of the considered clip. Concerning the position, in Video-encoded Point Clouds, this is generally obtained from the screen coordinates of a pixel, the view and projection matrices of the corresponding camera, and an associated auxiliary channel that encodes the distance of the point from the camera. By considering a unitary vector $\mathbf{r}$ that contains the world space ray connecting the center of the camera $\mathbf{c}$ to the pixel position and the distance d, the world space coordinates $\mathbf{x}$ can be easily computed as:

$$\mathbf{x} = \mathbf{c} + \mathbf{r} \cdot d \tag{1}$$

One of the main features of PBR, is that the irradiance map and the pre-filtered environment maps can be obtained from a 360°

High Dynamic Range Image (HRDI) of the environment. In this work, we will then consider this feature, by adding to each scene a specific 360° background image, used both to define the colors of the pixels of the display not covered by any point and to compute the maps required by the PBR algorithm. This job is done by the *miss shader* which is used every time raytracing misses one of the point cloud points.

3.1 Acquisition of the Point Cloud.

Acquisition from a 3D scene. When the source of the clip is generated with computer animation techniques, the required data can be sampled directly from the scene in the 3D authoring software. If meshes are rendered with materials following the *Principled Bidirectional Reflectance Distribution Functions* (BRDF), which is currently one the most widely used technique in 3D productions [5,33], most of the parameters are already available and can be obtained with limited changes to the rendering pipelines, that can be easily scripted in most of the authoring tools. For example, in Blender [34], the distance from the camera and the normal vector directions can be produced as render passes for compositing purposes. The Roughness and Metalness can be instead produced with a specific material, starting from the values used by the principled shader. For what concerns the 360° (HDRI) of the environment, this can be generated using a panoramic camera and conventional render layering techniques, commonly used for compositing.

Acquisition from a real-world take. When creating an animated point cloud from live footage, several cameras are required, creating a setup similar to the one shown in Figure 2. Cameras are divided into *blocks* and placed in a circular pattern around the focus of the scene. Each camera block is composed of two units: a *depth sensing camera*, capable of measuring the distance of the points as well as their color, and a *fish-eye HDRI camera*. The two cameras are oriented in opposite directions. The depth-sensing camera points toward the center, and it is used to create the point cloud corresponding to the focus of the scene. The fish-eye HDRI camera captures the background. In particular, by using several 180^o fish-eye cameras and exploiting stitching techniques, a 360^o degree video of the environment can be obtained, excluding the focus of the scene. The depth images sensed by the cameras can be used to create a mesh topology of the considered object, from which the normal maps can be derived. Moreover, *material capture* techniques, based for example on Convolutional Neural Networks or other artificial intelligence algorithms, can be used to infer the roughness and metalness properties of the surfaces, like, e.g., in [35].

3.2 Layout and Encoding. The approach investigated in this paper considers a single stream of data that encodes both the animated point cloud of the focus and the 360^o HDRI video of the surroundings, as a set of synchronized video streams and meta-data. In particular, it builds up over the encoding technique proposed by the MPEG-VPCC standard. We start considering only the focus object, i.e. the one encoded as an animated point cloud. Frames are partitioned into *Patches*, each one containing a set of adjacent pixels, as seen by one of the depth-sensing cameras of the scene. Patches include a 16×16 block of pixels, all belonging to the view of the same camera. However, not all the pixels inside one block might be useful for the final point cloud: the accuracy of some points could be better encoded by a different camera, and can thus be ignored. To discern between the points taken into consideration in a single patch, a *mask channel*, similar to a conventional *alpha* channel, is added: it describes the occupancy of the frames, associating a white color to the points to be considered and a black color to pixels that do not generate points in the final cloud. This masking procedure allows the algorithm to assign arbitrary colors to the pixels not used in a patch, in a way that improves the

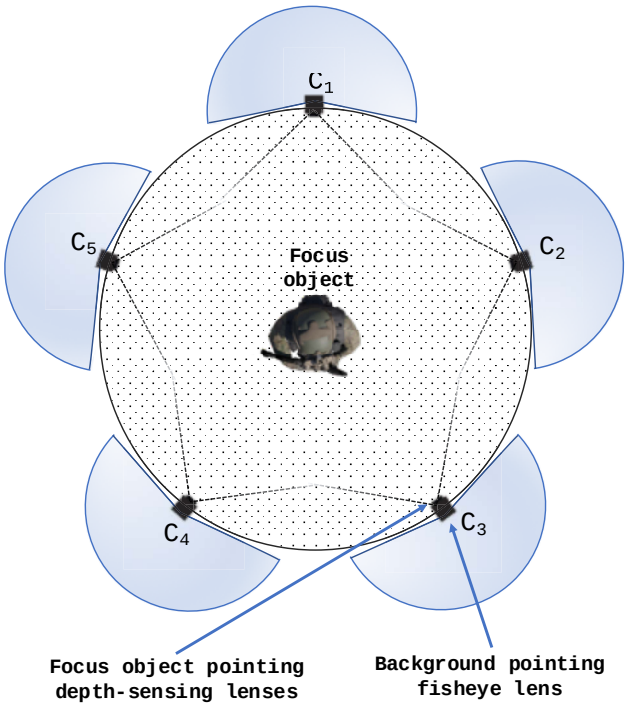


Fig. 2 Schema of the capture setup. C_n cameras are placed around the focus object.

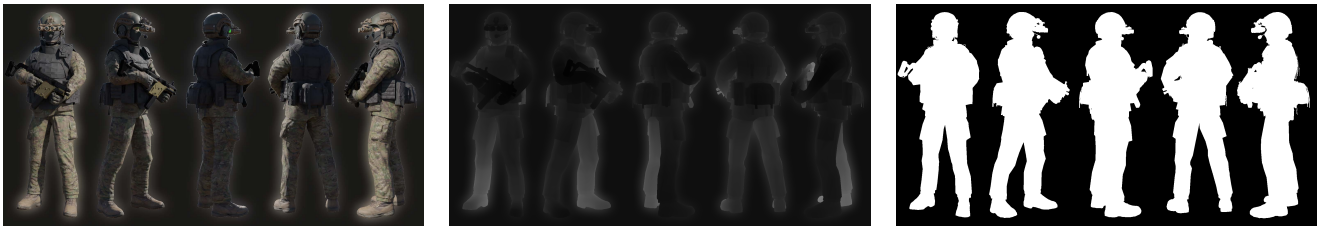
compression of the block [15]. The meta-data included with the frames encodes both the position and optical characteristics of the considered cameras and defines the image blocks associated with each of them.

The current proposal for animated point clouds considers only the three streams shown in Figure 3. Point colors (Figure 3(*a*)) and depth information (Figure 3(*b*)) share the same pixels and blocks distribution over the frame. They are thus characterized by the same camera matrix and cropped by the same occupancy (Figure 3(*c*)) mask. The example reported in Figure 3 shows the result where five different views of the same subject are patched in a single frame. The three channels have very different encoding requirements. The main color (Figure 3(*a*)) is a standard RGB channel, while the depth channel (Figure 3(*b*)) is monochromatic, but it should be encoded with higher precision to reduce the error when deriving the positions of points. The occupancy mask channel (Figure 3(*c*)) is essentially a binary channel, for which a lossless coding technique should be used to reduce errors in the output model.

To extend the current workflow, the four extra channels shown in Figure 4 should be added to support PBR. In particular, the four channels use the same camera and pixel layout defined in Figure 3 for the color and depth information. The first channel, shown in Figure 4(*a*), encodes the normal vector directions to the points of the cloud. As done in conventional *Object Space Normal Maps*, the direction, represented by a unit vector $\mathbf{n}$, is mapped to the RGB domain to define the color $\mathbf{y}_n$ of the corresponding pixel in the normal map. . The mapping function is defined according to the following equation:

$$\mathbf{y}_n = (\mathbf{n} + |1,1,1|)/2 \tag{2}$$

Roughness (Figure 4(*b*)) and Metalness (Figure 4(*c*)) are encoded as simple monochromatic channels, while the environment is encoded as a standard panoramic view image, following the *Equirectangular coordinates* (Figure 4(*d*)). Normal maps can be encoded using conventional RGB image coding techniques, while roughness and metalness are standard monochromatic channels. To



(*a*) Main Color (*b*) Depth (*c*) Occupancy

Fig. 3 An animation frame, showing the channels traditionally used by mpg-vpcc.

(*a*) Normal Map (*b*) Roughness

(*c*) Metalness (*d*) 360^o environment

Fig. 4 An animation frame, showing the proposed additional channels to support PBR rendering. In addition to the three channels defining the main color, depth and occupancy (Figure 3), the normal map (a), the roughness (b), the metalness (c) and the surrounding environment (d) have been added.

improve rendering quality, the environment map should instead be encoded using an HDRI video technique.

3.3 Decoding and Rendering. While acquisition, patch layout and encoding occur at the source, the viewer has to perform both decoding and rendering. Decoding extracts the information to generate the point cloud and renders it using PBR. In particular:

(1) Positions of the points are evaluated for each pixel where the occupancy mask is white, using the camera descriptions in the meta-data, using Equation 1.
(2) Normal vector directions $\mathbf{n}$ are computed starting from the color $\mathbf{y}_n$ by inverting Equation 2 as follow:

$$\mathbf{n} = 2\mathbf{y}_n - |1,1,1| \tag{3}$$

(3) Albedo colors, roughness and metalness are taken directly from the corresponding channels.
(4) The irradiance map and the pre-filtered environment maps, can be computed by filtering the 360° background image. Since these filtering procedures can be too heavy from a computational point of view to be performed in real-time, the usual approximations based on the MIP-MAP levels of the image can be performed. In particular, the irradiance map can be approximated with the $2\times2\times2$ MIP-MAP level, and roughness can be used for selecting the level to consider the most appropriate specular reflection.

The final pixel color can then be computed using the conventional techniques for implementing real-time PBR with image-based lighting. Such techniques, exploiting the pre-filtered maps, can be extremely efficient and easily performed in real-time on standard hardware.

4 Comparative Study

This section provides the results obtained by rendering a point cloud using PBR and image-based lighting. Since the focus of this work is on evaluating the quality of the PBR and image-based lighting rendering and comparing the results obtained with conventional point-cloud rendering techniques, the artifacts and errors caused by the lossy encoding and decoding process will be neglected. In particular, we start from an already decoded point cloud, presented as a set of vertices characterized by a set of attributes. We also simplify the processing of materials, considering *uniform surfaces*, characterized by a constant roughness and metalness value. These visual results will be compared with point clouds rendered using acquisition time colors only. To conduct the experiments, an animated point cloud is used, where each frame is stored in a different file. The specific model used comes from the *basketball_player* 20 seconds sequence provided by [36], which is a collection of 600 files stored in *.ply* format acquired at 30 frames per second.

As a pre-processing step, the normals have been checked on all the available point clouds: depending on the approach used to acquire and compress the point cloud, normals could have already been estimated or not. For example, in the case of V-PCC, normal estimation happens during patch generation. It is also possible to estimate them after scan-time, with techniques where normals are estimated from images acquired by depth cameras [37]. For our test case, the normals have been computed using a k-th nearest neighborhood algorithm, exposed by *MeshLab*[38]. The algorithm

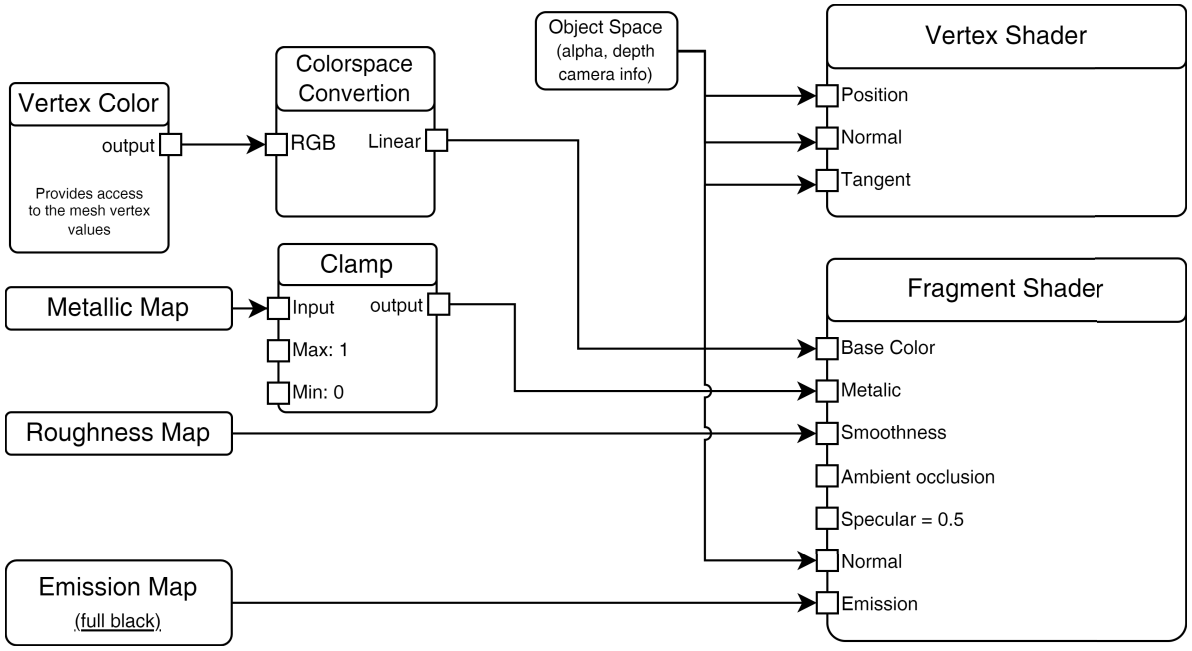


Fig. 5 Graph of the PBR shader able to render a point cloud model in a physically plausible way.

estimates the normal vector of each point in models without information about triangle connectivity. For each point of the cloud, it gets its k nearest neighbors and estimates the tangent plane with a principal component analysis. Regarding the choice of the k parameter defining the size of the neighborhood to fit the tangent plane, low values are usually suggested to better capture the local curvature of the model. If k is too large, points from different surfaces might be included in the neighborhood, resulting in distortions of the plane and badly estimated normals, especially around corners. MeshLab suggests values between 10 and 30 for a good estimation [38]: in this work, k has been set to 15 as it generates good results with the PC under consideration.

As the rendering engine, *Unity3D*[3] has been used. However, since Unity does not inherently handles *.ply* assets, we used a third-party importer called Pcx[4]. Once imported, point clouds are treated as a standard 3D mesh with just vertices and colors (i.e. no faces) that can be rendered using the standard components provided by Unity (i.e. the *MeshRenderer*), along with any custom shader. The Universal Render Pipeline (URP) with Shader Graph support has been enabled. We use the URP to create a shader able to visualize the point clouds with the same colors collected at scan time. Since the built-in Unity render pipeline uses a linear color space with sRGB light intensity, while URP applies linear light intensity, and since vertex colors stored in the used clouds are RGB, they look extremely bright. This is corrected by adding a Colorspace Conversion node to the basic shader, which handled the conversion to linear space. This first shader will be used as a baseline to compare the results of our proposed method.

A second *PBR* shader is then created as shown in Figure 5, implementing the improved pipeline proposed in this paper. The albedo is set as for the basic unlit shader, with the color given by vertices and colorspace conversion to linear space. The fragment normal space is set in Object space. Metalness and smoothness are set as properties to allow easier and faster customization from the Unity inspector and linked to a Clamp node to limit the values in the range [0, 1]. As default, both values have been put to 0, which is a reasonable value to realistically render the available basketball player model. There is no interest in showing an emissive color from the used model, so the emission color is set as full black. Ambient occlusion is also set as a property and will be tweaked accordingly to the scene in which the model is placed to better simulate the lighting intensity.

Figure 6 shows the same point cloud rendered with the PBR shader and the basic shader. In this scene, a low-intensity direct light is used, a spotlight is placed above the point clouds and the HDRI is the default skybox from Unity. Even in this basic scene, a notable increase in realism can be appreciated between the two rendering techniques application. Many small details, less evident with the basic shader, can be easily distinguished when the same model is rendered with the proposed method, e.g. the wrinkles of the player's t-shirt. Most importantly, the models on the left of Figures 6a and 6b look different in colorization with respect to the right models in the same Figures: this is due to the different incoming light direction and intensity, a detail that is not caught by the standard shaders (models on the right of Figures 6a and 6b).

In Figure 7, the method is tested in a different setting, inspired by the typical room used to test global illumination algorithms. In this scene, light-emitting planes are used as the only light sources of the scene. As it can be seen, the point cloud rendered with the PBR shader (the left one in Figure 7) is influenced by the light emitters, reacting to environment lighting when placed nearby a light source, while there is no effect on the unlit point cloud.

In Figure 8, a more realistic test scene is used. The scene represents a basketball court[5]. For this scene, a daylight setting is used, in the form of a background 360 degrees equirectangular video recorded on a beach in California. A single, directional light is added and oriented consistently with the position of the sun in the video. As already pointed out, when using the PBR shader more details become distinguishable, but even more interestingly, it is able to react when occluded by other objects in the scene, becoming darker due to the projected shadow, while the behavior of the model lit using the basic shader doesn't correspond to what is expected in this condition.

In Figure 9, the basketball court used in the previous comparison test is illuminated using a 360° video recorded in an empty city square at night. Taking advantage of the court model structure, a spotlight is placed on each of the four lamps on the court sides,

[3] https://unity.com/; LTS version 2020.3.22f1; [Online; accessed September 29, 2023]

[4] https://github.com/keijiro/Pcx; [Online; accessed September 29, 2023]

[5] https://skfb.ly/6UIT8

(*a*) (*b*)

Fig. 6 The same point cloud rendered with the *PBR* shader, left in each figure, and with the *basic* shader, right in each figure. (a) Low-intensity direct light illuminating both point clouds and (b) Spot light placed above the point clouds.

Fig. 7 Visual comparison between the PBR rendering (left) and standard rendering (right) of the same point clouds.

(a) (b)

Fig. 8 Comparison between the models when occluded the grate surrounding the basket court. (a) the point cloud model is rendered with PBR shader; (b) rendered using the basic shader.

Fig. 9 Visual comparison between the PBR rendering (left) and standard rendering (right) of the same point clouds when they face one of the spotlights.

oriented towards the center of the field. A strong point light is placed at the source position of each spotlight to better simulate the glowing effect of light emitted from lamps. It can be immediately seen that the point cloud model colored using the basic shader does not really fit with the environment, being very bright.

5 Conclusions and Future Works

In this paper, we demonstrated the potential of PBR use in the rendering of animated point clouds to achieve better visual realism. The approach proposed in this paper differs from the traditional pipeline as it takes into account animated point clouds and exploits four additional channels compared to the standard proposed by the MPEG community for point cloud compression. As per the pre-processing time, in addition to the ones required by the traditional pipeline, only the maps related to the four additional channels (normal, roughness, metalness and 360^o environment maps) have to be generated. Those can be easily generated with any offline 3D rendering software (e.g. Blender). The tests performed using Unity as rendering engine showed how, with PBR shading, a point cloud better integrates with the surrounding environment in any lighting condition, by casting shadows due to direct lighting and self-occlusions as well as revealing many color details captured from the surroundings, as coded by 360^o videos used for backgrounds. As point clouds are becoming more popular, using PBR for their rendering would be beneficial for many applications, going from simple model visualization to immersive experiences in Virtual or Mixed Reality.

Nevertheless, the proposed method has some limitations, especially regarding the computational efficiency. The animation of the player we used for rendering is a collection of points clouds each representing one frame of total movement. This required, for each rendering pass, the removal of the previous frame model before drawing the current frame, resulting in an overhead of operations that has to be optimized. Since each model of the *basketball_player* sequence comprises about three million vertices with color and normal vectors information (about 80MBs per model) and since at runtime the models are stored together in a list, even using just 20 frames of the animation results in occupying about 1.5GBs of RAM. This will strongly impact the rendering performances and is a challenge that has to be addressed. Another promising area open to future investigation is the possibility of applying our method to a video stream, thus to real-time capture, where the animated point cloud focus object is recorded using several connected cameras in a different environment than the one of the final rendering. The 360° video used as the background, in this case, could also come from live capture or being pre-recorded. The level of complexity for such an operation is surely increased and may require machine learning approaches to estimate and synchronize features like the lights, the colors, the depth and the normals estimation of the points cloud on-the-fly, from different cameras.

References

[1] Javaheri, A., Brites, C., Pereira, F., and Ascenso, J., 2021, "Point Cloud Rendering After Coding: Impacts on Subjective and Objective Quality," IEEE Transactions on Multimedia, **23**, pp. 4049–4064.

[2] Han, X.-F., Jin, J. S., Wang, M.-J., Jiang, W., Gao, L., and Xiao, L., 2017, "A review of algorithms for filtering the 3D point cloud," Signal Processing: Image Communication, **57**, pp. 103–112.

[3] Ipsita, A., Duan, R., Li, H., C, S., Cao, Y., Liu, M., Quinn, A. J., and Ramani, K., 2023, "The Design of a Virtual Prototyping System for Authoring Interactive VR Environments from Real World Scans," Journal of Computing and Information Science in Engineering, pp. 1–18.

[4] Kivi, P. E. J., Makitalo, M. J., Zadnik, J., Ikkala, J., Vadakital, V. K. M., and Jaaskelainen, P. O., 2022, "Real-Time Rendering of Point Clouds With Photorealistic Effects: A Survey," IEEE Access, **10**, pp. 13151–13173.

[5] Pharr, M., Jakob, W., and Humphreys, G., 2018, *Physically based rendering: From theory to implementation, Third Edition*, Morgan Kaufmann, -.

[6] McAuley, S., Hill, S., Martinez, A., Villemin, R., Pettineo, M., Lazarov, D., Neubelt, D., Karis, B., Hery, C., Hoffman, N., and Andersson, H. Z., 2013, "Physically based shading in theory and practice," *ACM SIGGRAPH 2013 Courses*, ACM, doi: 10.1145/2504435.2504457.

[7] Richardt, C., Tompkin, J., and Wetzstein, G., 2020, "Capture, Reconstruction, and Representation of the Visual Real World for Virtual Reality," *Real VR – Immersive Digital Reality*, Springer International Publishing, pp. 3–32.

[8] Berger, M., Tagliasacchi, A., Seversky, L. M., Alliez, P., Guennebaud, G., Levine, J. A., Sharf, A., and Silva, C. T., 2016, "A Survey of Surface Reconstruction from Point Clouds," Computer Graphics Forum, **36**(1), pp. 301–329.

[9] Schwarz, S., Preda, M., Baroncini, V., Budagavi, M., Cesar, P., Chou, P. A., Cohen, R. A., Krivokuca, M., Lasserre, S., Li, Z., Llach, J., Mammou, K., Mekuria, R., Nakagami, O., Siahaan, E., Tabatabai, A., Tourapis, A. M., and Zakharchenko, V., 2019, "Emerging MPEG Standards for Point Cloud Compression," IEEE Journal on Emerging and Selected Topics in Circuits and Systems, **9**(1), pp. 133–148.

[10] Cao, K., Xu, Y., and Cosman, P., 2020, "Visual Quality of Compressed Mesh and Point Cloud Sequences," IEEE Access, **8**, pp. 171203–171217.

[11] Kronander, J., Banterle, F., Gardner, A., Miandji, E., and Unger, J., 2015, "Photorealistic rendering of mixed reality scenes," Computer Graphics Forum, **34**(2), pp. 643–665.

[12] Christensen, P., 2008, "Point-based approximate color bleeding," Pixar Technical Notes, **2**(5), p. 6.

[13] Debevec, P., 2006, "Image-Based Lighting," *ACM SIGGRAPH 2006 Courses*, Association for Computing Machinery, New York, NY, USA, p. 4–es, doi: 10.1145/1185657.1185686.

[14] Sun, T., Xu, Z., Zhang, X., Fanello, S., Rhemann, C., Debevec, P., Tsai, Y.-T., Barron, J. T., and Ramamoorthi, R., 2020, "Light stage super-resolution," ACM Transactions on Graphics, **39**(6), pp. 1–12.

[15] Graziosi, D., Nakagami, O., Kuma, S., Zaghetto, A., Suzuki, T., and Tabatabai, A., 2020, "An overview of ongoing point cloud compression standardization activities: video-based (V-PCC) and geometry-based (G-PCC)," APSIPA Transactions on Signal and Information Processing, **9**(1), pp. –.

[16] Zerman, E., Ozcinar, C., Gao, P., and Smolic, A., 2020, "Textured Mesh vs Coloured Point Cloud: A Subjective Study for Volumetric Video Compression," *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*, IEEE, Athlone, Ireland, pp. 1–6, doi: 10.1109/qomex48832.2020.9123137.

[17] Benedek, C., Majdik, A., Nagy, B., Rozsa, Z., and Sziranyi, T., 2021, "Positioning and perception in LIDAR point clouds," Digital Signal Processing, **119**, p. 103193.

[18] Wang, Q. and Kim, M.-K., 2019, "Applications of 3D point cloud data in the construction industry: A fifteen-year review from 2004 to 2018," Advanced Engineering Informatics, **39**, pp. 306–319.

[19] Scharstein, D. and Szeliski, R., 2002, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," International Journal of Computer Vision, **47**(1), pp. 7–42.

[20] Barbierato, E., Gribaudo, M., Iacono, M., and Piazzolla, P., 2018, "Second Order Fluid Performance Evaluation Models for Interactive 3D Multimedia Streaming," *Computer Performance Engineering*, Springer International Publishing, pp. 205–218.

[21] Kim, H., Kitahara, I., Kogure, K., and Sohn, K., 2006, "A Real-Time 3D Modeling System Using Multiple Stereo Cameras for Free-Viewpoint Video Generation," *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 237–249.

[22] Intel, 2023, "Intel RealSense," https://www.intelrealsense.com/, [Online; accessed September 29, 2023].

[23] Zhang, Z., 2012, "Microsoft Kinect Sensor and Its Effect," IEEE MultiMedia, **19**(2), pp. 4–10.

[24] 8i, 2023, "Studios," https://8i.com/, [Online; accessed September 29, 2023].

[25] Boyce, J. M., Dore, R., Dziembowski, A., Fleureau, J., Jung, J., Kroon, B., Salahieh, B., Vadakital, V. K. M., and Yu, L., 2021, "MPEG Immersive Video Coding Standard," Proceedings of the IEEE, **109**(9), pp. 1521–1536.

[26] Jang, E. S., Preda, M., Mammou, K., Tourapis, A. M., Kim, J., Graziosi, D. B., Rhyu, S., and Budagavi, M., 2019, "Video-Based Point-Cloud-Compression

Standard in MPEG: From Evidence Collection to Committee Draft [Standards in a Nutshell]," IEEE Signal Processing Magazine, **36**(3), pp. 118–123.
[27] Akhtar, A., Gao, W., Li, L., Li, Z., Jia, W., and Liu, S., 2022, "Video-Based Point Cloud Compression Artifact Removal," IEEE Transactions on Multimedia, **24**, pp. 2866–2876.
[28] CAO, C., 2020, "What's new in Point Cloud Compression?" Global Journal of Engineering Sciences, **4**(5), pp. –.
[29] ISO/IEC MPEG (JTC 1/SC 29/WG 7), 2021, "G-PCC Codec Description V2," .
[30] Sheng, X., Li, L., Liu, D., and Xiong, Z., 2022, "Attribute Artifacts Removal for Geometry-Based Point Cloud Compression," IEEE Transactions on Image Processing, **31**, pp. 3399–3413.
[31] Wang, J., Zhu, H., Liu, H., and Ma, Z., 2021, "Lossy Point Cloud Geometry Compression via End-to-End Learning," IEEE Transactions on Circuits and Systems for Video Technology, **31**(12), pp. 4909–4923.
[32] Debevec, P., 2006, "Image-based lighting," *ACM SIGGRAPH 2006 Courses on - SIGGRAPH '06*, ACM Press, doi: 10.1145/1185657.1185686.
[33] da Silva Nunes, M., Nascimento, F. M., Miranda, G. F., and Andrade, B. T., 2021, "Techniques for BRDF evaluation," The Visual Computer, **38**(2), pp. 573–589.
[34] Blender Development Team, "Blender 3.1.0," https://www.blender.org
[35] Deschaintre, V., Aittala, M., Durand, F., Drettakis, G., and Bousseau, A., 2018, "Single-image SVBRDF capture with a rendering-aware deep network," ACM Transactions on Graphics, **37**(4), pp. 1–15.
[36] Xu, Y., Lu, Y., and Wen, Z., 2017, "Owlii Dynamic human mesh sequence dataset," https://mpeg-pcc.org/Downloads/Owlii/Owlii.zip, [Online; accessed September 29, 2023].
[37] Molnar, S., Kelenyi, B., and Tamas, L., 2021, "ToFNest: Efficient normal estimation for time-of-flight depth cameras," *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, IEEE, Montreal, BC, Canada, pp. 1791–1798, doi: 10.1109/iccvw54120.2021.00205.
[38] Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G., 2008, "MeshLab: an Open-Source Mesh Processing Tool," *Eurographics Italian Chapter Conference*, V. Scarano, R. D. Chiara, and U. Erra, eds., The Eurographics Association, doi: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.