AI in Telecommunication

QuietXO
Faculty of Electrical Engineering and Informatics
Technical University of Košice
Letná 9, 042 00 Košice
Email: rposa01@gmail.com

Abstract

Telecommunication is becoming an inseparable aspect of our everyday lives. With the rise in computation power and research in the field of artificial intelligence, we are getting closer to making this experience even better. Using photo-realistic avatars in virtual or augmented reality should make the experience more natural and is a promising path for achieving authentic face-to-face communication in 3D over remote physical distances.

1. Introduction

Photo-realistic Telepresence in Virtual Reality describes a technology for enabling authentic communication over remote distances so that each communicating party feels the genuine co-location presence of the others. [1]

With the advent of modern Virtual Reality headsets, there is a need for improved real-time computer graphics models with enhanced immersion. Traditional computer graphics techniques are capable of creating highly realistic renders of static scenes but at a high computational cost. These models also depend on physically accurate estimates of geometry and shading model components. When high precision estimates are difficult to acquire, degradation in perceptual quality can be pronounced and difficult to control and anticipate. Finally, the photo-realistic rendering of dynamic scenes in real-time remains a challenge. [6]

We can solve the issue by scaling the rendering in the virtual reality telepresence to the number of people in it and computing only the visible pixels. Some of the recent works in neural rendering are:
- The deferred neural rendering [3] Section 2.1
- The neural point-based graphics [4] Section 2.2
- The implicit differentiable rendering [5] Section 2.3

These implementations use a neural network to compute pixel values in the screen space and not the texture space, thus computing only visible pixels. In all of these implementations, we assume that the scene is static or that our viewing distance and perspective are not totally free in the 3D space.

This would be a huge problem in our case, and therefore there is another approach Pixel Codec Avatar (PiCA) see Section 3, which is a deep generative model of 3D human faces that achieves a state of the art reconstruction performance while being computationally efficient and adaptive to the rendering conditions during execution. [1] This model is a combination of two core ideas:
1) A fully convolutional architecture for decoding spatially varying features
2) Rendering adaptive per-pixel decoder

This makes multi-person telecommunication possible and that's thanks to both techniques being integrated via a dense surface representation which is learned in a weakly-supervised manner from low-topology mesh tracking over training images. While we are still a bit limited by the number of people that we are able to render in real-time into one scene, PiCA model is still a lot smaller than the state-of-art baseline model and is able to generate 5 avatars on a single Oculus Quest 2.

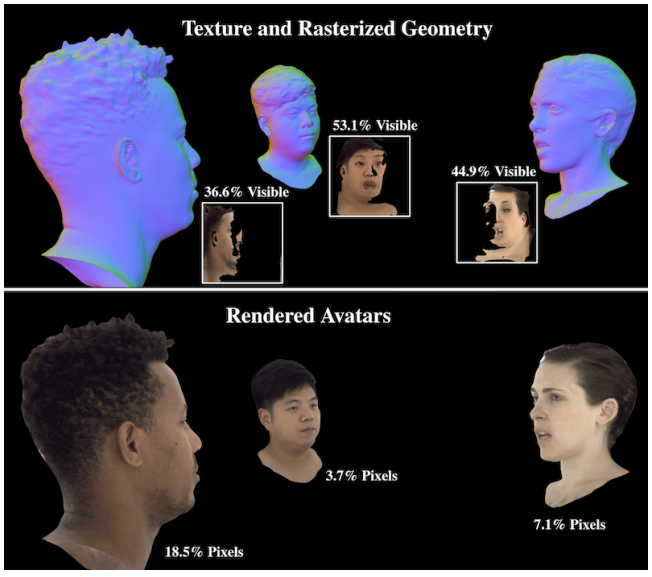

Figure 1. A multi-person configuration for a teleconference in virtual reality. In the upper side of the image, we can see that at any given moment and from any viewing angle we can see roughly half of the head, while in the lower part of the image we see how many pixels are occupied by the head.

2. Related Work

2.1. The deferred neural rendering

This approach learns the deferred rendering process directly from real images without needing any expert knowledge and allows us to resynthesize novel views of an object, edit scenes and synthesize animations. It proposes a unified neural rendering pipeline based on video footage of a target object we estimate its 3D geometry. This geometry is coarse and not suited to be processed by the standard graphics pipeline to achieve photo-realistic images. It's also able to learn specific components of the rendering pipeline such that we can generate new images based on course geometry. Given a 3D model, we parametrize its surface using a UV atlas, then using the standard rasterizer, we draw corresponding UV maps of the object to the image space.

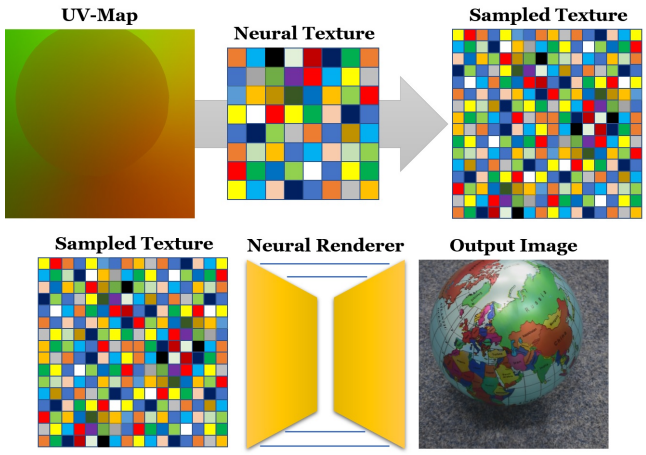

Figure 2. Overview of the neural rendering pipeline: Given an object with a valid UV-map parameters and an associated Neural Texture map as input, the standard graphics pipeline is used to render a view-dependent screen-space feature map. The screen space feature map is then converted to photo-realistic imagery based on a Deferred Neural Renderer. [3]

This UV map is input to the method, and the pipeline consists of learnable neural textures and a learnable renderer. Instead of classical textures, this neural texture can contain learned features which are interpreted by the rendering network. Afterward, the UV maps from the rasterizer are used to sample from the neural textures and a small unit is used to translate these features to color values. This pipeline can be trained end to end given ground truth pairs of UV and color maps from the original object. Since the neural textures are attached to the object surface we are able to edit a scene. Given the input sequence and the 3D reconstruction, we can easily duplicate objects in a scene, and using this modified geometry we rasterize the corresponding UV map, which is an input to our rendering pipeline.

Neural textures are also a powerful tool that can enable animation synthesis. To this end, we can demonstrate it on facial reenactment given a source and a target actor, where we reconstruct the facial geometry and transfer the expressions from the source to the target mesh. The modified UV map, as well as a background image from the target video, will be used as input for the rendering approach.

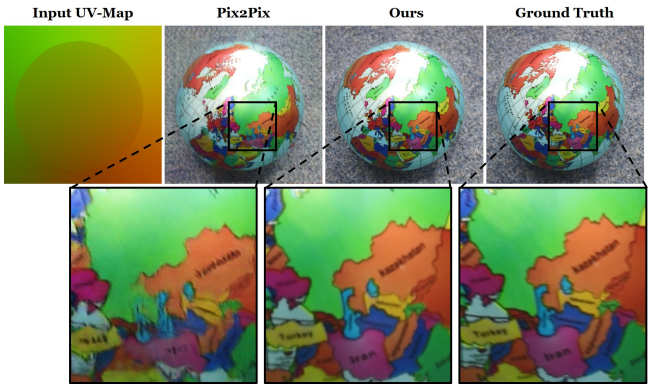

Figure 3. Comparison to the image-to-image translation approach Pix2Pix. As can be seen, the novel views synthesized by the deferred neural rendering is higher quality, e.g. less blurry and results are close to the ground truth.

The rendering quality is dependent on the resolution of the texture. A texture at a single resolution might be too coarse or too fine, which leads to over-fitting and under-sampling during training. Because of this, we are better off using a hierarchy of textures resulting in better sampling behavior, particularly for higher texture resolutions. The number of training images also influences the rerendering quality. In particular, a view-dependent effects are not well captured with a reduced set of images, such as specular highlights slowly degrade.

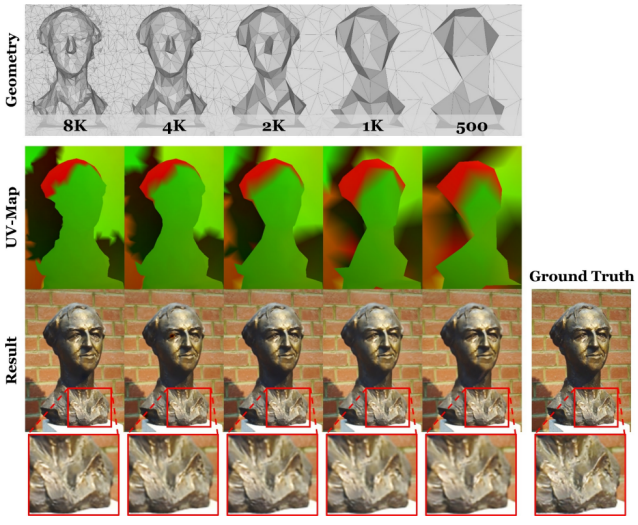

Figure 4. The resolution of the underlying geometry proxy. Using quadric edge collapse we gradually reduce the number of triangles of the geometry from 8000 to 500. Even with the lowest resolution, a photo-realistic image can be generated. The MSE measured on the test sequence increases from 11.068 (8K), 11.742 (4K), 12.515 (2K), 18.297 (1K) to 18.395 for a proxy mesh with 500 triangles (MSE w.r.t. color channels in [0, 255]) [3]

2.2. The neural point-based graphic

This approach allows you to capture diverse static 3D scenes from image data and render them from new viewpoints. It also uses point clouds to represent scene geometry and a raw point cloud as the geometric representation of a scene. Such point clouds can be obtained by stereo-matching registered RGB image sets. Alternatively, we can obtain them by fusing depth maps from registered RGBD videos. Even with a noisy and incomplete underlying point cloud, this approach can generate complete and realistic views for novel viewpoints. To facilitate rendering from new viewpoints it assigns each point a neural descriptor of low dimensionality, where the descriptor contains information about local photo-metric and geometric properties. (For the experimentation were used eight-dimensional descriptors.)

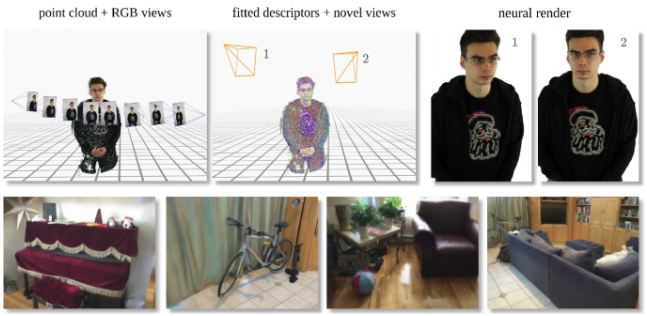

Figure 5. Given a set of RGB views and a point cloud (top-left), this approach fits a neural descriptor to each point (top-middle), after which new views of a scene can be rendered (top-right). The method works for a variety of scenes including 3D portraits (top) and interiors (bottom). [4]

To render a new view, we can project points onto the novel view at several resolutions using descriptor values as pseudo color. Afterward, we use a rendering convolutional network to transform the multi-resolution rendering into a photo-realistic image, and to capture the appearance of a scene we will fit the descriptors and the rendering network parameters to a given set of views. The fitting uses back-propagation and minimizes the loss between the rendered views and the ground truth frames.

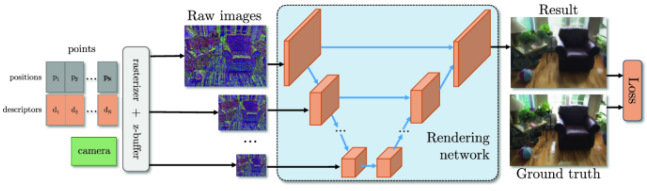

Figure 6. An overview of our system. Given the point cloud P with neural descriptors D and camera parameters C, we rasterize the points with z-buffer at several resolutions, using descriptors as pseudo-colors. We then pass the rasterizations through the U-netlike rendering network to obtain the resulting image. Our model is fit to new scene(s) by optimizing the parameters of the rendering network and the neural descriptors by back-propagating the perceptual loss function. [4]

When compared to the neural rendering system, which uses a mesh rather than a point cloud, is a geometric proxy. In general, both methods achieve similar results, but the neural point-based approach performs better whenever meshing fails. For example, thin object parts are a problem for the mesh (Figure 7), whereas a point-based solution does not have this problem. Although when the images in the training set suffer from inconsistent exposure or have inaccurate registration point-based method can suffer from temporal flickering. This effect can be alleviated by applying temporal smoothing and spatial anti-aliasing to the rendering of the point cloud before they are processed by the rendering network.

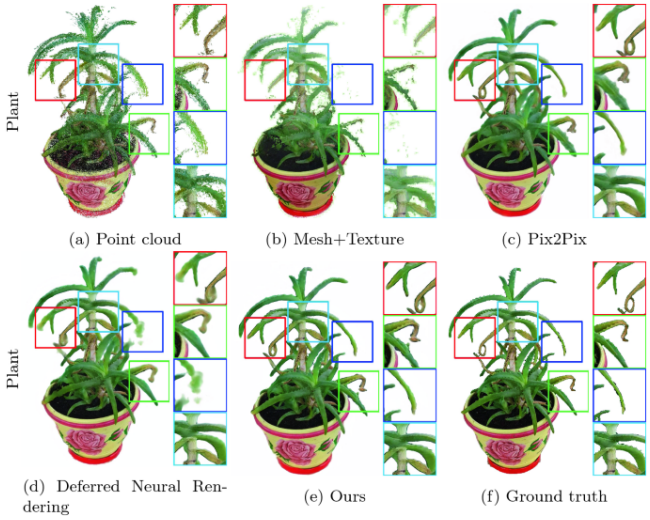

Figure 7. Comparative results on the holdout frame from the 'Plant' scene. The neural point-based method better preserves thin parts of the scene, while the deferred neural rendering method blurs or leaves out this detail.

Among other objects, we assess the performance of this method on human portraits. We can combine two scenes into a single one and for that, we fit both of them while keeping the coefficients of the rendering network the same, we then merge the point clouds after rough geometric alignment. The rendering network can then be applied to the combined point cloud.

2.2.1. The accelerated neural point-based graphic. (NPBG++) [2] It is built upon the Neural Point-Based Graphics method and improves it in several significant ways. Fundamentally, it lifts the limitation of pre-scene optimization by directly predicting the neural features from the input images and processing the source views one at a time by an alignment to a canonical orientation, followed by feature extraction, obtaining features that will be used to update the state of the point cloud. The input image alignment ensures, that the features are consistent since the feature extractor is not rotation-equivariant by default.

Once all of the views have been processed, we finalize the aggregation procedure which will be described next. The procedure results in view-dependent descriptors which can be used to synthesize new views following the rendering

scheme from Neural Point-Based Graphics. The resulting image is rotated from the canonical orientation to the target view alignment, yielding the final output. The system is end-to-end trainable, resulting in a model that can easily adapt to new scenes, without performing additional optimization.

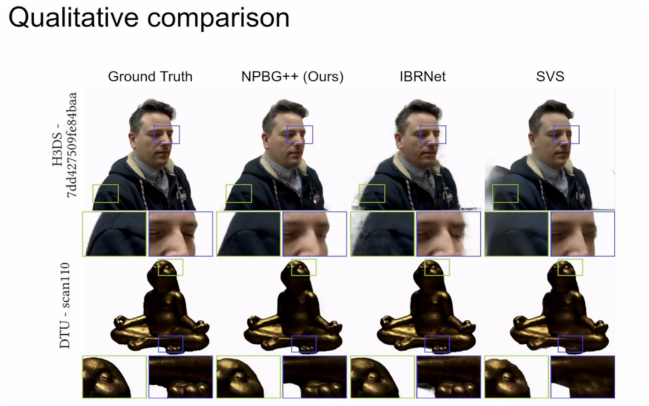

Figure 8. Comparative results on multiple methods of rendering, among which we can see that NPBG++ is the superior one in terms of details and sharpness.

2.3. The implicit differentiable rendering (IDR)

This is a neural network architecture, that can learn three unknowns (Geometry, Appearance, Camera parameters) simultaneously and can produce high-fidelity 3D surface reconstruction, by disentangling geometry and appearance, learned solely from masked 2D images and rough camera estimations. We achieve this by processing these three unknowns with a differentiable rendering system. That means, simulating the rendering of the scene from a given view, and comparing the rendered image to the original image.

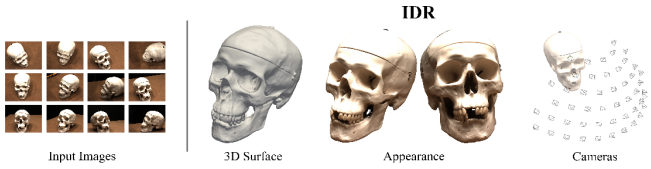

Figure 1: We introduce IDR: end-to-end learning of geometry, appearance and cameras from images.

Figure 9. Learning of geometry, appearance and cameras from images.

The key challenge is how to make the renderer a function of the geometry, that is independent of the object material or the scene lighting. A recent work by Niemeyer et al. [8] introduced such a differential renderer model that can represent arbitrary texture, but cannot handle reflectance and lighting effects, nor can it handle noisy cameras. Concurrently to this work, NeRF [7] used neural rendering for novel views synthesis from a set of images with known cameras, where differently unlike IDR their method does not produce a surface reconstruction of the scene's geometry, rather than a 3D density field. In this work, we reconstruct an implicit surface together with a surface light field that is separated from the geometry and we can handle both exact and noisy camera information. The geometry is represented as the zero level set of a neural network f, which model each 3D point its sign distance function to the shape. Given a learnable camera position and some fixed image pixels, we would like to produce differentiable RGB values. The camera orientation and pixel define a viewing direction, and we can trace the first intersection of the viewing ray with the implicit surface.

The first step is to represent the intersection point and its normal as differential functions of the implicit geometry and the camera parameters. We implement it by simply composing the neural network with a fixed linear computation at its entrance and another at its output.

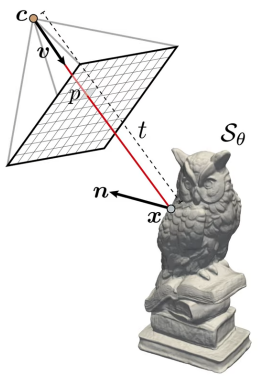

Figure 10. Notation.

Secondly, we want to approximate the color of the pixel, determined by the radiance reflected from the surface to the camera. Our interest is that the renderer will not memorize any part of the geometry or the viewing cameras. Therefore, we suggest representing the light field as a function of the surface position, surface normal, and viewing direction. This is implemented using a second neural network to output RGB values. Incorporating the normal and the viewing direction is necessary for 3D reconstruction that is decoupled from the appearance and the cameras.

This claim can be explained using this simple example (Figure 11). A rendered without normal will produce the same light estimation in those two cases (Figure 11 *a, b*), therefore can result in a renderer that compensates over geometry properties. A renderer without viewing direction will produce the same light estimation in those two cases (Figure 11 *a, c*), wherein real-life the scene appearance is likely to be view-dependant.To complete the IDR model, we can use the global feature vector of the geometry, to get our final radiance approximation. Which is then compared to the ground truth pixel color, to simultaneously train the model's parameters.

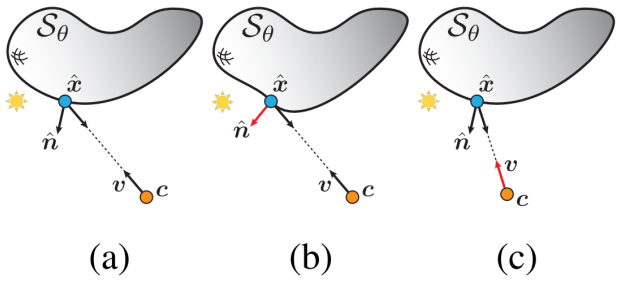

Figure 11. Neural renderers without n and/or v are not universal.

3. Pixel Codec Avatar (PiCA)

The Pixel Codec Avatar is a conditional variational autoencoder (VAE) where the latent code describes the state of the face (e.g., facial expression) and the decoder produces realistic face images.

PiCA renders realistic faces by decoding the color of each rasterized or raycast pixel using a shallow SIREN that takes as input a local expression code, the 3D coordinates in object space, and the positional encoded surface coordinates. This particular combination allows the feature dimensions and network size to remain small and computationally efficient while retaining image fidelity. The local expression codes and geometry are decoded using fully convolutional architectures from a global latent code and the viewing direction, and require only small resolutions of 256x256. Learnable components are supervised on multiview images, depth, and tracked coarse mesh.

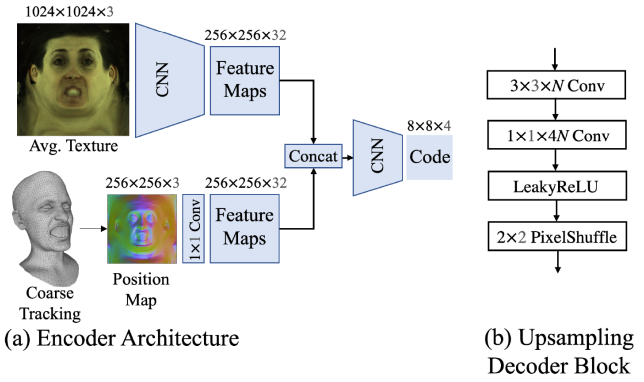

Figure 12. (a) The encoder. (b) The basic block in the geometry decoder and expression decoder.

Encoder (Figure 12*a*) encodes the average texture, computed over unwrapped textures of all camera views, and a tracked mesh into a latent code. Note this tracked mesh is coarse, containing 5K vertices, and does not contain vertices for tongue and teeth. We only assume availability of such coarse mesh for training because face tracking using dense mesh over long sequences with explicit teeth and tongue tracking is both challenging and time consuming. Requiring only coarse mesh in training makes our method more practical. In Lombardi et al., the 3D coordinates of mesh vertices are encoded using a fully connected layer and fused with texture encoder; in contrast, we first convert the mesh into a position map using a UV unwrapping of the mesh. Joint encoding of the geometry and texture is then applied, and the final code is a grid of spatial codes, in our case an 8x8 grid of 4 dimensional codes.

Geometry Decoder takes the latent code as input and decodes a dense position map describing face-centric 3D coordinates at each location. The architecture is fully convolutional, and the basic building block (Figure 12*b*). We convert the position map to a dense mesh by sampling at each vertex's UV coordinates, and rasterize it to determine visible pixels. In our experiments, the position map is 256×256 and the extracted dense mesh has 65K vertices.

Expression Decoder uses the latent code and the viewing direction to decode a low resolution, view-dependent map of local codes. It consists of the decoder block (Figure 12*b*) and the output map is 256×256 in our experiments.

Pixel Decoder decodes the color at each facial pixel given p. Specifically, rasterization determines whether a screen pixel corresponds to a visible mesh point, and, if so, the triangle id and barycentric coordinates of the mesh point. This allows us to compute the encoding inputs p from the expression map, the vertex coordinates, and the UV coordinates of the triangle. Inspired by the pixel-wise decoding of images in Sitzmann et al., the pixel decoder is designed as a SIREN. However, we use a very lightweight network by design, with 4 layers and a total of 307 parameters. We utilize effective encoding in u to produce facial details with such a light model.

While neural networks and MLPs in particular can represent functions of arbitrary complexity when given sufficient capacity, lightweight MLPs tend to produce low-frequency outputs when given smoothly varying inputs. Thus, given only the smooth face-centric coordinates and surface coordinates as input, a lightweight pixel decoder tends to produce smooth output colors for neighboring pixels, leading to a loss of sharpness in the decoded image. Instead, we encode information about such spatial discontinuities at the input of the Pixel Decoder using two strategies: a low resolution local expression code z for dynamics, and a learned non-parametric positional encoding u of surface coordinates for detail. These complement the mesh coordinate input x, which encodes face-centric *xyz* coordinates using a two-layer SIREN.

4. Conclusion

We still got a way to go, but if we keep up this pace, it shouldn't take long to achieve even greater results in this field. With the grow of faster computation units, we are able to get the results quickly, but we have to work on some kind of optimization as well, since we are running close to the end-point of how much we can push it.

Acknowledgment

I would like to thank my deadline mode for putting up with me and helping me finish stuff that somehow ends up being pushed to the last moment.

References

[1] Shugao Ma, Tomas Simon, Jason Saragih, Dawei Wang, Yuecheng Li, Fernando De la Torre and Yaser Sheikh. Pixel Codec Avatars *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 1, 5

[2] Ruslan Rakhimov, Andrei-Timotei Ardelean, Victor Lempitsky and Evgeny Burnaev. NPBG++: Accelerating Neural Point-Based Graphics *(CVPR)*, 2022. 3

[3] Justus Thies, Michael Zollh¨ofer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Trans. Graph.*, 38(4), 2019. 1, 2

[4] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. *arXiv preprint arXiv:1906.08240*, 2019. 1, 3

[5] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Ronen Basri, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. 2020. 1, 4

[6] Stephen Lombardi, Jason Saragih, Tomas Simon, and Yaser Sheikh. Deep appearance models for face rendering. *ACM Trans. Graph.*, 37(4), 2018. 1

[7] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*, 2020. 4

[8] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. *arXiv preprint arXiv:1912.07372*, 2019. 4