Pacific Graphics (2021)
M. Okabe, S. Lee, B. Wuensche, and S. Zollmann (Editors)

Neural Proxy: Empowering Neural Volume Rendering for Animation

Zackary P. T. Sin, Peter H. F. Ng, and Hong Va Leong

The Hong Kong Polytechnic University, Hong Kong, China

Abstract

Achieving photo-realistic result is an enticing proposition for the computer graphics community. Great progress has been achieved in the past decades, but the cost of human expertise has also grown. Neural rendering is a promising candidate for reducing this cost as it relies on data to construct the scene representation. However, one key component for adapting neural rendering for practical use is currently missing: animation. There seems to be a lack of discussion on how to enable neural rendering works for synthesizing frames for unseen animations. To fill this research gap, we propose neural proxy, a novel neural rendering model that utilizes animatable proxies for representing photo-realistic targets. Via a tactful combination of components from neural volume rendering and neural texture, our model is able to render unseen animations without any temporal learning. Experiment results show that the proposed model significantly outperforms current neural rendering works.

CCS Concepts

• ***Computing methodologies*** → *Computer graphics; Machine learning;*

1. Introduction

Photo-realistic rendering that mirrors reality is potentially a holy grail for computer graphics. With many recent modern advances, we can see that many movies and digital games are able to render scenes that sometimes blur the boundary of reality. However, most of the scenes are expensive to acquire as they require intensive human effort to handcraft the scene representation. Neural rendering is a recent growing topic that proposes teaching a neural machine to generate photo-realistic imageries. It is a promising candidate for reducing the cost of realistic rendering. Although current neural rendering works such as NeRF [MST*20] and neural texture [TZN19] are able to produce impressive results, most of them are only able to render static scenes or video scenes where the dynamics are recorded. There are works that can synthesize for novel motion, but they are limited to data-rich domains such as human motion. For many interactive applications (e.g. games, augmented reality apps), however, animations are an important part of the experience. To elevate neural rendering to a practical photo-realistic solution for digital content making, how to enable it for synthesizing animated frames is an important research question to address.

A typical computer graphics pipeline uses a 3D mesh for where to render, and a texture for what to render. Could neural rendering imitate this intuitive pipeline more closely? Our investigation on this idea paves the way to *neural proxy*. Instead of a 3D model with a texture, we have a neural proxy with a neural texture. The neural proxy determines where the photo-realistic target will render while the neural texture provides a high-level description on how to render the target's surface realistically. Another key idea in our method is a specific integration between neural volume rendering and neural texture. The result is a model that can render photo-realistic frames of an unseen animation for a target given only its static pose via a small collection of images (Figure 1). To summarize, our contributions are as follows:

- *Neural proxy*, a novel method that empowers animations for neural rendering by representing photo-realistic targets as manipulatable proxies.
- A tactful integration between neural volume rendering and neural texture that enables rendering of animated frames without seeing any animations in training.
- Results that show neural proxy's improvement over existing methods and illustrate the model's ability in synthesizing photo-realistic animations.

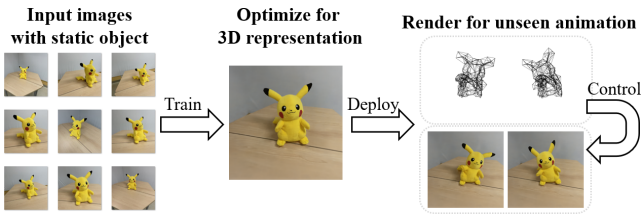

Figure 1: *Neural proxy is a neural rendering technique that can render frames for an unseen animation via an animatable proxy. It only requires training views with a shared static pose.*

DOI: 10.2312/pg.20211384

2. Related Work

Image-based neural rendering refers to works that manipulate with image representations. An early work in this regard is pix2pix [IZZE17], a conditional imagery GAN [GPAM*14]. It can accept a UV map or a segmentation mask for generating a photo-realistic output. Neural texture [TZN19] is, arguably, one piece of earlier work that explicitly models components of a computer graphics pipeline to a neural rendering work. Instead of a typical texture, say albedo (RGB) texture, [TZN19] proposed to sample from a learned neural feature map. The learned features are then fed to a neural renderer for producing photo-realistic results. However, a limitation of this work is that it has difficulty rendering frames for unseen animations if the renderer learns from static objects and has not seen them moving.

Neural volume rendering is a neural adaptation to the classical volumetric rendering in computer graphics. A marked beginning of this line of work is neural volume [LSS*19]. It samples a voxel grid of neural features with ray marching. This voxel grid of embedded features is somewhat similar to the one in [STH*19]. Neural radiance field (NeRF) [MST*20] is a follow-up work to neural volume. NeRF proposed that a sampler can be directly fed with a continuous Cartesian coordinate to produce a particle. As the content is tied to a coordinate, it is difficult to manipulate for animations.

It is possible to render animated results with neural texture [TZN19]. The renderer, however, must be given animated training samples to achieve this, like [RTH*20] which assumes a data-rich human motion domain. Similarly, [LXZ*19] synthesizes different human actors via a driving video. There is a body of works that aims to introduce a temporal component to neural volume rendering. Nerfies [PSB*20] uses a multilayer perceptron (MLP) to produce a deformation field such that a moving particle will be sampled from the same reference coordinate. [PCPMMN20] also proposes a similar scheme to model a video's dynamics. Instead of using a MLP for deformation, [LNSW21] uses scene flow. [YLSL21] enables handling of moving objects without supervised training. However, although the dynamic models mentioned can handle temporal information, they cannot be used for general animations as they rely on latent interpolation for generating new animated content. This design makes it difficult to specify the desired animation. Furthermore, if the content to generate is not between any known/learned latent points, it will be impossible to generate.

3. Building Blocks

In this section, we will be discussing two major works of neural rendering: neural texture and NeRF. One of the key ideas of our method is to tactfully combine these two works in such a way that their inherent modelings that hinder animation synthesis are avoided.

Neural texture [TZN19] provides localized surface appearance information such that the texture's neural feature is fed to a neural renderer for rendering. This method can be formulated as $I = f_{NR}(f_{Bi}(U;\Omega_{Tex});\theta_{NR})$, where I is the rendered image; f_{NR} is the neural renderer and its parameters are θ_{NR}; f_{Bi} is a bilinear sampler; U is the UV map; and Ω_{Tex} is the neural texture. A problem exists with respect to θ_{NR} for animated content as it implies that the synthesized content I relies on what is learned. If θ_{NR} cannot be generalized for different manipulations of the surface, it may fail for rendering animated frames as there will be many possible manipulations for a new animation.

NeRF [MST*20] is a current state-of-the-art model for neural volume rendering. It utilizes a ray of particles to aggregate the color for a pixel on an image. The authors proposed that the particles could be generated by a sampler neural network S_{NeRF} such that, $(c,\sigma) = S_{NeRF}(x,d)$, where c and σ are the color and the density of a particle respectively, x is a point in the continuous Cartesian coordinate and d is the current view direction for modeling view-dependent lighting. By accumulating the colors and the densities of particles along a ray r, the color of a pixel, $C(r)$ can be computed. This accumulation is approximated by the method detailed in [MST*20]. With a ray of particles $x_i = r(t_i) = o + t_i d$ where x_i is the Cartesian coordinate of the i-th sampled particle, t is a distance sampled uniformly, and o and d are the origin and the direction of a ray r respectively, we can render a pixel with $T_i = exp(-\sum_{j=1}^{i-1}\sigma_j\Delta t_j)$, $\hat{C}(r) = \sum_{i=1}^{N_s} T_i(1-exp(-\sigma_i\Delta t_i))c_i$, where $\hat{C}(r)$ approximates $C(r)$, $(c_i,\sigma_i) = S_{NeRF}(x_i,d)$, $\Delta t_i = t_{i+1} - t_i$ and N_s is the number of sampled particles along r. In practice, [MST*20] also introduced a positional encoding which manipulates in a higher dimension. As NeRF's input is based in a Cartesian coordinate, it is not easy to manipulate for animating the content.

4. Methodology

The motivation of our work is to empower animations for neural rendering. The core idea is to inject neural proxies into a scene such that each neural proxy represents its photo-realistic counterpart. By animating these proxies, animated photo-realistic frames can be rendered. Our work mimics the typical computer graphics pipeline for neural rendering (Figure 2*a*).

To achieve our goal, we use a volumetric renderer and a neural proxy wrapped with a neural texture for localizing particles. This design circumvents the need for training a renderer with animated frames which may be intractable to provide. By animating the proxy, localized particles will move with it, resulting in an animation. Our proposed method starts with the construction of a proxy that relies on shell maps [PBFJ05], a classical computer graphics technique. For modeling localized surface appearance, we wrap the proxy with a neural texture. For rendering a scene with animatable targets and static surroundings, we use a mixture of a neural proxy sampler and a static sampler for particles sampling.

4.1. Proxy Construction

The construction of a proxy starts by retrieving a reference geometry, which can be done by a structure-from-motion technique such as COLMAP [SF16], or manually (e.g. use a low-resolution icosphere for a high-fidelity football). Then, the volume of the proxy that specifies how far the proxy is influencing the photo-realistic result is constructed. This volume needs to enclose the photo-realistic target such that the relevant particles are fully contained. In computer graphics, a classical way to construct this volume is shell maps [PBFJ05], which we propose here to adapt for neural rendering. The volume can be constructed by viewing the coarse geometry

as a "thick surface", which can be further understood as an outward surface extruded from the original surface to form a shell. It is easy to see that by extruding a triangle, a prism is formed, thereby transforming from an area to a volume. The original coarse geometry is defined as $\bar{M}$ and we assume that it is populated with vertices v_i with smoothed normals n_i such that $\bar{M} = \{v_i\}_{i=1}^{N_v}$, where N_v is the number of vertices in $\bar{M}$. The extrusion is done so that there is an outward surface $\overrightarrow{M}$ and an inward surface $\overleftarrow{M}$. We denote the notion of outward (inward) with the arrow $\leftarrow$ ($\rightarrow$). Their vertices can be computed by $\overrightarrow{v_i} = v_i + n_i \cdot \overrightarrow{\varepsilon}$ and $\overleftarrow{v_i} = v_i - n_i \cdot \overleftarrow{\varepsilon}$, where $\overrightarrow{\varepsilon}$ ($\overleftarrow{\varepsilon}$) is the tuned offset parameter for the outward (inward) surface under the conditions that $\overrightarrow{\varepsilon}, \overleftarrow{\varepsilon} >= 0$, $\overrightarrow{\varepsilon} + \overleftarrow{\varepsilon} > 0$. It should be set such that the shell will enclose the target object. To generate a volume that guarantees no intersection between prisms, one could use the method stated in [CVM*93]. In practice, by using reasonable $\overrightarrow{\varepsilon}$ and $\overleftarrow{\varepsilon}$, such method is not necessary for neural proxy.

We will wrap a neural texture around the proxy later. So, we need to define the UV coordinates for the extruded surfaces. For the original 3D mesh $\bar{M}$, we assume that there is a UV coordinate u_{2D} for each vertex. As the proxy's volume (or shell) contains the particles for rendering, we need some method to tell the 3D location. This can be achieved by lifting the 2D UV coordinate to 3D with a proxy depth component w such that we have $\overrightarrow{u_i} = (u_{2D}, \overrightarrow{w})$ and $\overleftarrow{u_i} = (u_{2D}, \overleftarrow{w})$, where $\overrightarrow{u_i}$ is the UV coordinate of $\overrightarrow{v_i}$. The new third dimension component w is referring to how deep a point is within the proxy's shell. For easier conceptual understanding, we choose $\overrightarrow{w} = 1$, which indicates the outward shell, and $\overleftarrow{w} = 0$ which indicates the inward shell. However, it is simply a label to feed to a learnable network. The labels need not have an explicit range as long as $\overrightarrow{w}$ and $\overleftarrow{w}$ are different. After the extrusion, we now have M which includes $\overrightarrow{w}$ and $\overleftarrow{w}$ such that $M = \{\overrightarrow{v_i}\}_{i=1}^{N_v} \cup \{\overleftarrow{v_i}\}_{i=1}^{N_v}$. To sample particles via the neural texture, we need to map between the shell space and the texture space. To achieve this, the Barycentric coordinate system can be used. The prisms, however, should be first broken into three tetrahedra with the method detailed in [PBFJ05].

4.2. Proxy Shell Sampling

The proxy's shell is the basic building component for the neural proxy to sample features from its neural texture. Recall from Section 3 that neural volume rendering requires sampling in the Cartesian coordinate such that a sampler will produce the color c and the density σ given a location. For our proposed neural proxy, this process stays similar with a Cartesian point x to be given to the neural proxy sampler S_p such that $(c, \sigma) = S_p(x)$ (the view component d is dropped for simplicity), which produces a two-valued output. The key difference between the samplers from NeRF and neural proxy is what actual information is used for processing. NeRF uses global information as input (a point in the scene's space) while neural proxy uses local information as input (the feature sampled from a neural texture). Before sampling the neural texture, we need to first travel from the Cartesian coordinate to the Barycentric coordinate. The function that translates between the coordinates is defined as $\varphi_\tau = f_{Cart \to Bary}(x; \tau)$, where φ_τ is tetrahedron τ's Barycentric counterpart of x. Suppose we wish to convert a point with the Cartesian coordinate x to its Barycentric coordinate $\varphi_\tau = (\varphi_A, \varphi_B, \varphi_C, \varphi_D)$ for a tetrahedron τ consisting of vertices ABCD, the following volumes within the tetrahedron will need to be first computed, $V_A = \frac{1}{6}(x - x_B) \cdot ((x_D - x_B) \times (x_C - x_B))$, $V_B = \frac{1}{6}(x - x_A) \cdot ((x_C - x_A) \times (x_D - x_A))$, $V_C = \frac{1}{6}(x - x_A) \cdot ((x_D - x_A) \times (x_B - x_A))$, $V_D = \frac{1}{6}(x - x_A) \cdot ((x_B - x_A) \times (x_C - x_A))$ and $V_{ABCD} = \frac{1}{6}(x_B - x_A) \cdot ((x_C - x_A) \times (x_D - x_A))$, where x_A is the Cartesian coordinate of the vertex A. The Barycentric components can then be computed by comparing the volumes such that we have $\varphi_A = \frac{V_A}{V_{ABCD}}$, $\varphi_B = \frac{V_B}{V_{ABCD}}$, $\varphi_C = \frac{V_C}{V_{ABCD}}$ and $\varphi_D = \frac{V_D}{V_{ABCD}}$. It is important to check which sample $x \in X$ is within which tetrahedron, or any at all. The following equation can be used to check if a sample is within a tetrahedron τ, $X_\tau = \{x \in X | \varphi_A, \varphi_B, \varphi_C, \varphi_D \geq 0 \wedge (\varphi_A, \varphi_B, \varphi_C, \varphi_D) = f_{Cart \to Bary}(x; \tau)\}$ which can also help us find all points X_τ that lie within the volume of τ. To find all points that lie within any of the tetrahedra, $X_{M_\tau} = \bigcup_{i=1}^{N_\tau} X_{\tau_i}$, where $M_\tau = \{\tau_i\}_{i=1}^{N_\tau}$ and N_τ is the number of tetrahedra, can be used instead. On the other hand, we can also derive a method to find τ_x, the tetrahedron that contains x. It can be done by $\tau_x = \{\tau \in M_\tau | x \in X_\tau\}$. Finding τ_x is important as we need to know which tetrahedron x is in for sampling the neural texture. To find the UV coordinate of a point, given its Barycentric coordinate, it can be computed with $u = f_{Bary \to UV}(\varphi_\tau; u_\tau) = \varphi_A u_A + \varphi_B u_B + \varphi_C u_C + \varphi_D u_D$, where u_A is the UV coordinate of the vertex A and $u_\tau = \{u_A, u_B, u_C, u_D\}$. The UV is now retrievable for each location in a Cartesian coordinate.

4.3. Neural Proxy

Neural proxy is a key component of our proposed method. It is a proxy representation of the photo-realistic target. The proxy aforementioned is only a volume and does not contain the appearance information. We propose using a neural texture to wrap around the proxy to specify its fine surface information. Here, a learned network S_{Tex} that mimics a neural texture is used to sample a neural feature ϕ given a UV coordinate such that $\phi = S_{Tex}(u; \theta_{Tex})$. Then, to sample the particles of a neural proxy, it can be done with $(c, \sigma) = \hat{S_p}(\phi; \theta_p)$, which we will later simplify as $S_p(x)$ for the sake of clarity. Now, we are able to use a Cartesian point x to sample a neural feature ϕ and feed it to $\hat{S_p}$ for rendering a particle.

It is now possible to use neural proxies to model a scene. However, at this stage, it will require us to construct a proxy for every object in the scene. Although this is possible, there may be scenarios where we want to have two rendering routes (Figure 2*c*): one that uses the neural proxy sampler for important objects that we want to animate; and, the other that uses a scene sampler for modeling static elements. To this end, we also propose mixing the neural proxy sampler S_p with a neural scene sampler S_c. The architecture of $\hat{S_p}$ and S_c are identical to NeRF (i.e. S_{NeRF}).

Recall that the key to neural volumetric rendering is to utilize a set of particles to compute the color of an image pixel. These particles are located in a set of points x in Cartesian coordinates such that $x \in X$. As we are going to sample from the two samplers, S_p and S_c, we need to separate X such that a proxy set X_p is fed to S_p, and a scene set X_c to S_c, where $X = X_p \cup X_c$. Instead of finding the entire X_p which may be very expensive to compute, we simply find the sections of rays that intersect with X_p. For efficiency, the section for a ray r is here represented as the convex set X_r bounded by the closest point and the farthest point where r intersects with M (Figure 2*b*). Using the triangle intersection method f_δ from [MT97],

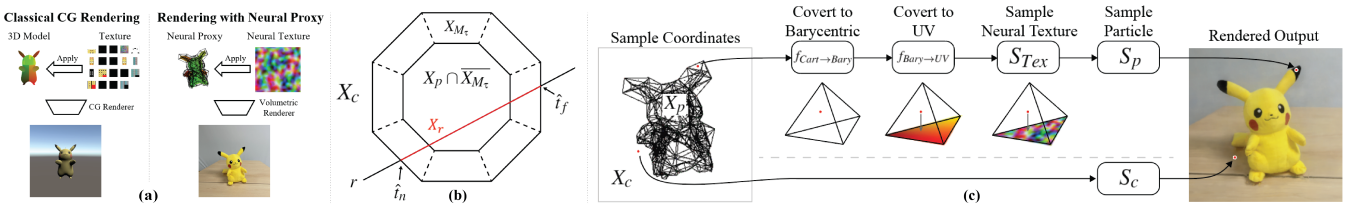

Figure 2: *(a) The intuition of neural proxy is to imitate a typical computer graphics pipeline. The proxy mirrors a 3D model while the neural texture is a neural replacement for a typical texture (e.g. albedo). (b) Each ray r is checked for its convex set X_r where particles are considered to be within the proxy (red). (c) Neural proxy's pipeline. A sample is handled by the proxy sampler S_p and the scene sampler S_c separately depending on whether it is within the proxy i.e. X_p or outside the proxy i.e. X_c. The neural texture is used to sample particles for local surface appearance.*

we can get a three-valued function $(\hat{t}, u_\delta, v_\delta) = f_\delta(r, \delta)$, where δ is the triangle in question with three vertices; $\hat{t}$ is the distance such that $r(t) = o + \hat{t}d$ is where the intersection will be, if occurred; and u_δ, v_δ are the Barycentric coordinate of where the intersection occurred. In [MT97], $u_\delta, v_\delta \geq 0, u_\delta + v_\delta \leq 1$, need to be true such that the ray r has intersected with the triangle δ. Now, the convex set where r intersects with M can be found with the following,

$$\begin{aligned} &D_\delta(r) = \{(\hat{t}, u_\delta, v_\delta) | (\hat{t}, u_\delta, v_\delta) = f_\delta(r, \delta) \wedge \delta \in M_\delta\} \\ &D = \{\hat{t} | (\hat{t}, u_\delta, v_\delta) \in D_\delta(r) \wedge u_\delta, v_\delta \geq 0 \wedge u_\delta + v_\delta \leq 1\} \\ &\hat{t}_n = \min D, \hat{t}_f = \max D, \end{aligned} \tag{1}$$

where $D_\delta(r)$ is a three-valued set for all triangles, D is a set of distances to where intersections occurred and M_δ denotes the set of all the triangles in M. Then, the samples on the ray r that we will feed to S_p are therefore,

$$X_r = \begin{cases} \emptyset & \text{if } D = \emptyset \\ \{r(\hat{t}) | \hat{t}_n \leq \hat{t} \leq \hat{t}_f\} & \text{otherwise.} \end{cases} \tag{2}$$

To render an image, however, will require processing a batch of rays. The full sampled proxy set for rendering an image (or partially) is $X_p = \bigcup_{r \in R} X_r$, where R is a batch of rays. Finally, we will have the following for mixing the two samplers,

$$(c, \sigma) = S(x) = \begin{cases} S_p(x) & \text{if } x \in X_p \cap X_{M_\tau} \\ S_c(x) & \text{if } x \in X_c \\ (0, 0) & \text{otherwise.} \end{cases} \tag{3}$$

The three conditions describe the different spaces that the sample is in (Figure 2*b*). For the first condition, $X_p \cap X_{M_\tau}$ (which can be simplified as X_{M_τ}) specifies the volume in which a sample will be both within the proxy and its shell. Thus, it can be sampled by S_p. For the second condition, it can be seen that it is referring to the volume outside the proxy, therefore S_c is used. For the last condition, it is referring to the volume that is within the proxy P ($x \in X_p$) but not within the shell M_τ ($x \notin X_{M_\tau}$). Since it is not within any of the tetrahedra, it cannot be sampled by S_p. With $S(x)$, the image can be rendered ray by ray by using $\hat{C}(r)$ described in Section 3 and the model can be trained with a photometric loss $L = \sum_{r \in R} ||\hat{C}(r) - C(r)||_2^2$, where $C(r)$ is the ground truth. As we have two samplers in the same scene, we may need to stabilize the learning. A distillation process can be used [RPLG21]. Figure 2*c* illustrates the pipeline.

We are now able to reproduce a scene with neural proxy. Recall that our goal is to animate objects for neural volume rendering. Since the proxy M is basically a mesh, it can be converted to a skinned mesh, which enables animations via a skeleton. When the animation skeleton moves, our neural proxy sampler will sample according to the change of volume. Thus, the animation will propagate to the particles and finally, to the rendered result (Figure 1).

5. Result

The main goal of neural proxy is to enable the generation of frames for unseen animations via neural rendering. To animate a scene with neural proxy, one would need to provide images with camera parameters (intrinsic and extrinsic) and a 3D reference mesh for the target to animate. These two requirements can usually be provided by software such as COLMAP [SF16] simultaneously. Alternatively, the reference mesh can be handmade instead of reconstructed. It is also assumed that the mesh is UV mapped which can be done automatically with most 3D modeling software. In our experiments, we visually checked that the proxy volume contains the target by adjusting suitable offset parameters $\overrightarrow{\epsilon}$ and $\overleftarrow{\epsilon}$.

To evaluate the effectiveness of the proposed model, the experiments need to show its ability to generate frames for unseen animations. Providing samples for real data is difficult as the animated poses of the target object need to be retrieved. Therefore, our experiments mainly rely on synthetic data. Specifically, three synthetic objects are prepared, a ball (80 polygon faces), a ninja (190 faces) and the armadillo (184 faces). For each object, a set of training images is provided to a model for learning the object representation such that it can synthesize the object. Each training image is an RGB rendering viewing the object at a randomized view and a randomized distance. The ball, the ninja and the armadillo target has 2000, 1000 and 1000 training images respectively. They all share the same static pose such that no animation (or temporal information) is present in the training data. Each object has its own set of testing animations. The unseen animations are only rendered as testing images for evaluation. The animations are ball-bounce (60 frames), ball-intersection (100 frames), ninja-walking (50 frames), ninja-capoeira (100 frames), armadillo-punch (50 frames) and armadillo-kick (50 frames). All images are rendered in 512×512. MSE, PSNR, Intersection over Union (IoU) and Learned Perceptual Image Patch Similarity (LPIPS) [ZIE*18] are used for evaluation. IoU is for measuring whether the shape of

the target could be replicated during animations while LPIPS is for measuring the visual fidelity.

There is a limited number of neural rendering works to compare with. Neural volume works, by default, are used for modeling static scenes (e.g. NeRF [MST*20]). Similarly, works such as Nerfies [PSB*20] that model a video are not practical to synthesize for animations as they do not have a direct mechanism to control the exact pose. This leaves us with image-based methods such as neural texture [TZN19] and pix2pix [IZZE17]. To our knowledge, these two models are the closest neural rendering techniques that can enable animations. As neural texture and pix2pix require 2D maps as input, a sequence of animated UV maps is provided to them while for neural proxy, an animated proxy M is provided frame by frame. All models have never seen any animations during training.

As shown in Table 1, our neural proxy model is able to outperform the others in almost all metrics. This outcome shows that our model yields a significant improvement over current neural rendering works in synthesizing frames for unseen animations. These results are also reflected qualitatively on Figure 4. For pix2pix, as it does not have an explicit mechanism to map the UV to some appearance feature, it generally performed poorer than the other two models. We can see that for ball-bounce and armadillo-kick, the ball's pattern and the armadillo's texture degrade as the two targets are animated. For neural texture, as its learned renderer has not seen the animated frames, it is expected that it will produce artifacts when the object's pose deviates from the static pose it has seen during training. In ball-bounce, although neural texture can maintain the pattern of the ball, a halo artifact will intensify as the ball deforms while in armadillo-kick, tearing artifacts (i.e. tearing holes) will start to appear as the character animates. The texture and the shape of the armadillo also gradually loses their fidelity. In contrast to the two models, the proposed neural proxy, in general, is able to produce clearer results with higher visual fidelity. The high IoUs and low LPIPSs of neural proxy further validate its success in rendering frames for unseen animations.

The enticing proposition of animating a real scene is one of the key motivations that drives our work on neural proxy. For a classical computer graphics pipeline, animating a realistic scene will involve a high-quality 3D model that may be expensive to acquire. With our work, this cost is reduced to taking some pictures and preparing the geometry. To investigate if our method is applicable for rendering novel poses for animations given a real scene, we have taken 51 images of a scene involving a stuffed toy. COLMAP is used for reconstructing a proxy geometry and retrieving camera parameters. We set the rendering resolution as 512×512. As shown in Figure 3, by manipulating the skeleton of the proxy, we are able to render animations with photo-realistic quality. To the best of our knowledge, we are the first to show how to synthesize frames of an unseen animation for a real scene via neural volume rendering.

6. Conclusion and Discussion

In this paper, we have discussed the current limitations of neural rendering works in synthesizing frames for unseen animations. Our solution to this research gap is neural proxy, which utilizes a proxy to represent a photo-realistic target. In addition, by tactfully combining components from neural volume rendering and neural texture, our model is able to render animations without learning from any temporal input. We have provided a set of experiments which shows that our model is able to synthesize frames for unseen animations with considerable improvement over current neural rendering techniques. We also synthesized for a real scene, showcasing our model's promise in animated photo-realistic rendering.

Although the proposed model improves upon current works, artifacts can still be seen with our model. One of our future directions is to utilize inpainting techniques to smooth out these artifacts. Speed is another limitation, but it is expected to be possible to bring the rendering speed to real-time for interaction (e.g. [RPLG21]). Furthermore, as the neural texture represents the surface appearance of a target, an exciting direction we are pursuing is to further utilize it for mesh and texture editing. We expect that with further investigation, neural proxy can be used as a building block for a real-time photo-realistic neural renderer that can be used in an array of real-world applications such as digital games, augmented reality, 3D photo editing and more.

Figure 3: *The real scene result. The proposed neural proxy is able to synthesize unseen animation photo-realistically. Our model has never seen the character animated before.*

References

[CVM*93] COHEN J., VARSHNEY A., MANOCHA D., TURK G., WEBER H., AGARWAL P., BROOKS F., WRIGHT W.: Simplification envelopes. In *SIGGRAPH* (1993). 3

[GPAM*14] GOODFELLOW I. J., POUGET-ABADIE J., MIRZA M., XU B., WARDE-FARLEY D., OZAIR S., COURVILLE A., BENGIO Y.: Generative adversarial nets. In *NeurIPS* (2014). 2

[IZZE17] ISOLA P., ZHU J.-Y., ZHOU T., EFROS A. A.: Image-to-image translation with conditional adversarial nets. In *CVPR* (2017). 2, 5, 6

[LNSW21] LI Z., NIKLAUS S., SNAVELY N., WANG O.: Neural scene flow fields for space-time view synthesis of dynamic scenes. In *CVPR* (2021). 2

[LSS*19] LOMBARDI S., SIMON T., SARAGIH J., SCHWARTZ G., LEHRMANN A., SHEIKH Y.: Neural volumes: Learning dynamic renderable volumes from images. *ACM TOG 38*, 4 (July 2019), 65:1–65:14. doi:10.1145/3306346.3323020. 2

[LXZ*19] LIU L., XU W., ZOLLHÖFER M., KIM H., BERNARD F., HABERMANN M., WANG W., THEOBALT C.: Neural Rendering and Reenactment of Human Actor Videos. *ACM Transactions on Graphics 38*, 5 (Oct. 2019), 139:1–139:14. URL: http://doi.org/10.1145/3333002, doi:10.1145/3333002. 2

[MST*20] MILDENHALL B., SRINIVASAN P. P., TANCIK M., BARRON J. T., RAMAMOORTHI R., NG R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV* (2020). 1, 2, 5

[MT97] MÖLLER T., TRUMBORE B.: Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools 2*, 1 (July 1997), 1–7. doi:10.1080/10867651.1997.10487468. 3, 4

Model	Ball-bounce MSE↓	PSNR↑	IoU↑	LPIPS↓	Ball-intersection MSE↓	PSNR↑	IoU↑	LPIPS↓
pix2pix [IZZE17]	8.494	19.502	0.837	0.0691	8.135	22.220	**0.956**	0.0506
Neural Texture [TZN19]	9.018	**19.567**	0.833	0.0643	4.693	25.772	0.916	0.0343
Neural Proxy (Ours)	**7.819**	19.480	**0.850**	**0.0583**	**2.658**	**26.979**	0.952	**0.0179**

Model	Ninja-walk MSE↓	PSNR↑	IoU↑	LPIPS↓	Ninja-capoeira MSE↓	PSNR↑	IoU↑	LPIPS↓
pix2pix	9.709	20.943	0.891	0.0498	8.957	21.002	0.885	0.0560
Neural Texture	9.646	21.883	0.912	0.0382	8.473	22.635	0.917	0.0390
Neural Proxy	**8.768**	**23.046**	**0.931**	**0.0327**	**7.720**	**23.640**	**0.929**	**0.0358**

Model	Armadillo-punch MSE↓	PSNR↑	IoU↑	LPIPS↓	Armadillo-kick MSE↓	PSNR↑	IoU↑	LPIPS↓
pix2pix	11.395	19.022	0.858	0.0864	10.335	20.453	0.904	0.0595
Neural Texture	8.950	24.039	0.938	0.0421	**9.320**	21.938	0.920	0.0503
Neural Proxy	**8.889**	**27.568**	**0.942**	**0.0289**	9.395	**24.455**	**0.937**	**0.0337**

Table 1: *Comparison of the performance from different models.*

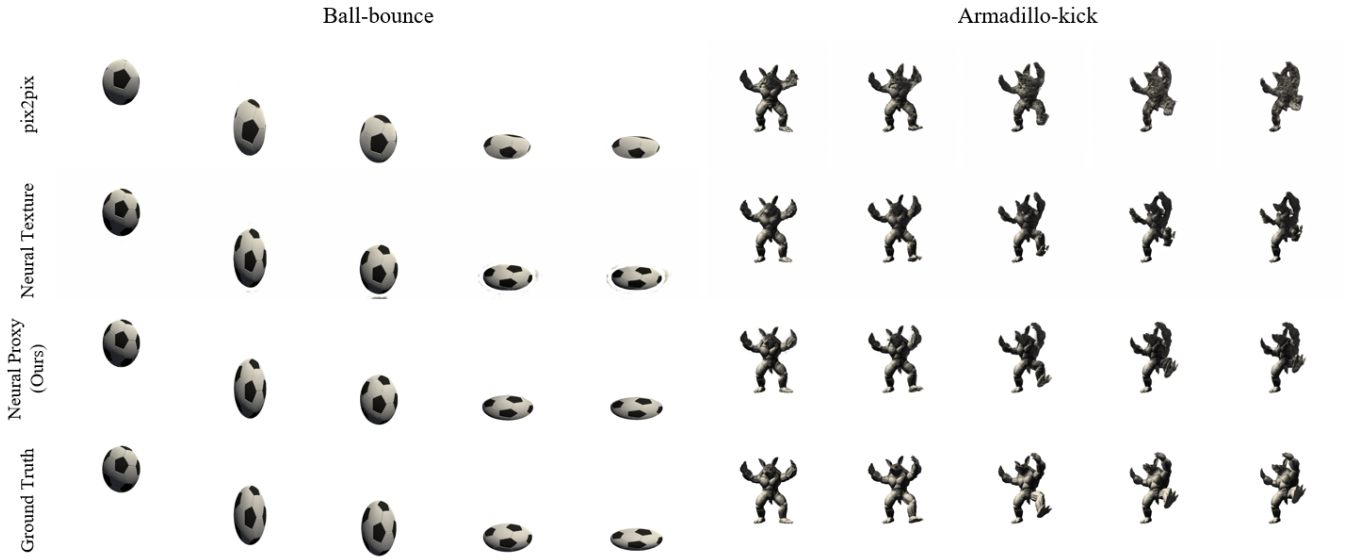

Figure 4: *The results of the ball-bounce animation and the armadillo-kick animation.*

[PBFJ05] PORUMBESCU S. D., BUDGE B., FENG L., JOY K. I.: Shell maps. *ACM TOG 24*, 3 (July 2005), 626–633. doi:10.1145/1073204.1073239. 2, 3

[PCPMMN20] PUMAROLA A., CORONA E., PONS-MOLL G., MORENO-NOGUER F.: D-NeRF: Neural Radiance Fields for Dynamic Scenes. In *CVPR* (2020). 2

[PSB*20] PARK K., SINHA U., BARRON J. T., BOUAZIZ S., GOLDMAN D. B., SEITZ S. M., MARTIN-BRUALLA R.: Deformable neural radiance fields. *arXiv preprint arXiv:2011.12948* (2020). 2, 5

[RPLG21] REISER C., PENG S., LIAO Y., GEIGER A.: Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. 2021. 4, 5

[RTH*20] RAJ A., TANKE J., HAYS J., VO M., STOLL C., LASSNER C.: Anr: Articulated neural rendering for virtual avatars. *arXiv preprint arXiv:2012.12890* (2020). 2

[SF16] SCHÖNBERGER J. L., FRAHM J.-M.: Structure-from-motion revisited. In *CVPR* (2016). 2, 4

[STH*19] SITZMANN V., THIES J., HEIDE F., NIESSNER M., WETZSTEIN G., ZOLLHÖFER M.: Deepvoxels: Learning persistent 3d feature embeddings. In *CVPR* (2019). 2

[TZN19] THIES J., ZOLLHÖFER M., NIESSNER M.: Deferred neural rendering: image synthesis using neural textures. *ACM TOG 38*, 4 (July 2019), 66:1–66:12. doi:10.1145/3306346.3323035. 1, 2, 5, 6

[YLSL21] YUAN W., LV Z., SCHMIDT T., LOVEGROVE S.: Star: Self-supervised tracking and reconstruction of rigid objects in motion with neural rendering. *arXiv preprint arXiv:2101.01602* (2021). 2

[ZIE*18] ZHANG R., ISOLA P., EFROS A. A., SHECHTMAN E., WANG O.: The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR* (2018). 4