Capturing View-Dependent Optics in 3D Feature Embeddings

Benjamin Choi
Stanford University
benchoi@stanford.edu

Abstract

We address the limited ability of 3D feature embeddings to correctly model view-dependent effects (e.g. specularities). We propose a new DeepVoxels architecture that learns a view-invariant 3D feature embedding in terms of geometry but retains view-dependence in optical effects in 2D renders of the object. Our general strategy is to isolate the modules responsible for implicitly modeling view-dependent effects in an imprecise way. Upon isolation, we inject these layers with explicit information regarded as necessary to precisely model view-dependent effects. Although we do not completely isolate the module responsible for implicit specularity rendering, we find that our approach is vastly superior to the baseline DeepVoxels model in 2D renders of specular objects and remains comparable to the baseline in non-specular object renders.

1. Introduction

Modern generative adversarial networks have proven to be effective in generating novel images using convolutional network architectures that use two-dimensional kernels. However, these networks struggle to have a learned understanding of three-dimensional spaces from which novel views could be synthesized. Recent work in the space of three-dimensional learning and novel view synthesis has largely struggled to achieve sufficiently high resolution when explicitly modeling three-dimensional representations of objects. DeepVoxels circumvents these issues by forgoing attempts to explicitly model objects in three-dimensions with three-dimensional supervised methods. Rather, it combines 3D feature representations with an occlusion network that attempts to learn which aspects of an object are occluded in each novel view.

However, DeepVoxels makes the assumption that objects are Lambertian, or have a "matte" diffuse reflecting surface. In reality, however, objects have view-dependent optical effects like specularities due to external light sources. It is not well understood how these specularities currently are or could be modeled, yet these view-dependent effects are essential to inferring novel views that appear as realistic as possible. Thus, our aim is to develop a more concrete, efficient understanding and strategy to model such view-dependent optical effects. As a concrete first step towards this goal, we aim to reduce the ambiguity and redundancy in the current DeepVoxels network. The input to our algorithm is an array of two-dimensional representations of an object from several viewpoints (thus allowing view-dependent optical effects to be conveyed). This is combined with the camera pose (i.e. from what perspective the picture was taken) to form the training corpus. We then use a combination of 2-D and 3-D U-Nets, gated recurrent units, and a new CNN-based rendering network to output a view-independent 3D voxel grid feature representation of the object. When rendered from differing viewpoints, our network will convey the appropriate view-dependent optical effects.

2. Related Work

Our work is related to the intersection of several active research areas, namely, 3D Reconstruction, Novel View Synthesis, and View-Dependent Rendering. While the original DeepVoxels model [10] addresses the former two areas in a highly proficient manner, it leaves the view-dependent rendering to the implicit capabilities of the model rather than explicitly attempting to model these effects. Thus, our aim is to integrate this area of research into DeepVoxels.

2.1. 3D Reconstruction

The mapping of two-dimensional images to some coherent, view-invariant three-dimensional feature embedding is central to our project. Current approaches like [6] attempt to learn a 3D point cloud representation of the object, emphasizing the accuracy of the overall three-dimensional structure rather than focusing on the appearance of surface features. In contrast, our project seeks to simply learn some feature embedding that allows for the best possible surface rendering. Unsupervised methods like [8] continue to be limited to emphasizing the volumetric or mesh component [13] of reconstructed 3D objects. Other strategies emphasize the ability to generalize to various perspectives via

encoder-decoder networks like in [15], but still excel mostly at rendering the overall shape rathern than detailed surface appearance.

2.2. Novel View Synthesis

The generation of novel views is precisely how we interpret our object, as we can gain a sense of the shape and position of an object in question by gaining perspective on the object from a wide variety of angles. Some approaches forgo deep learning methods in favor of purely mathematical interpolation to new views [14], while others use simple CNN based architectures [11] to make predictions regarding novel views. In practice, cGANs prove to be among the most effective approaches to generating images as close to "realistic" renders as possible [7], and a similar approach is indeed employed in our own experiments.

2.3. View-Dependent Rendering

Models like IGNOR [12] not only create 3D representations of objects, but also train entire networks to model desired qualities like view-dependent effects. Other models focus only on modeling materials and specularities using new modules like U-Nets that serve to encode and decode input data [2]. Still others make view-dependent effects the central focus of their experiments [3], aiming to create CNNs that can generate relatively high resolution reflectance maps and evaluate performance both on synthetic and real capture data.

3. Methods

At a high level, our version of DeepVoxels learns a 3D voxel grid representation that is invariant to viewpoint, but is capable of rendering view-dependent optical effects like specularities. Our model is a hybrid of the model architecture used in [10] as well as several custom modules and features that are hypothesized to be well-suited for understanding and improving performance in rendering view-dependent optical effects.

3.1. Training Input

On a general level, the input to our model is composed of a source view and a target view. The source view is a 128 by 128 RGB image, and is paired with camera pose information that conveys the cameras location in 3D space as well as its view angle. Each of these source views is associated with two target views. Each target view is selected randomly, one from the top five nearest neighbors, and the other randomly. During training, the model seeks to generate a predicted view for each of these target views. The "easier" (5-NN) and "harder" (All) target views serve to smoothen the optimization process, as the easier target view increases the likelihood that steps along the gradient will actually make meaningful progress, while the harder target view encourages the model to ensure that "bold" inferences from significantly different viewpoints match "risk-averse" inferences made from source views that were already closer to the target view (i.e. the "easier" case).

3.2. Model Architecture

The DeepVoxels network takes advantage of several powerful, modern deep learning modules in order to generate a high-performing 3D feature embedding. An understanding of these modules is essential to motivating our subsequent attempts to better understand and model specularities. Overall, our architecture is similar to the original DeepVoxels network, with several notable differences (in particular, rendering network and camera ray features) that aim to isolate the model's ability to appropriately render specularities.

3.2.1 Feature Extraction Network

The original DeepVoxels network first extracts features to a 128 x 128 grid via a 2D U-Net. U-Nets are composed of several downsampling and upsampling convolutional and transpose convolutional layers respectively. Downsampling serves to increase the amount of feature channels that can reasonably and efficiently be extract from an image, while upsampling serves to restore a sense of localization to the feature map. In this case, this localization is essentially one-to-one, as the input and output dimensions are identical (128 x 128). The implementation of U-Net we use consists of 5 downsampling and upsampling layers.

3.2.2 Integration Network

The integration network of DeepVoxels primarily deals with learning the 3D feature representation of the object, using gated recurrent units to control what features are integrated into the overall learned representation.

Firstly, 2D features must be lifted to a 3D feature volume. Conversion between 2D and 3D space requires variables describing the intrinsic and extrinsic factors of the camera pose [16]. More specifically, given a standard 3x4 intrinsic matrix K and 3x3 rotation R and 3x1 translation matrix T (origin of world center from in camera coordinates) we have that

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K[R\,T] \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} \tag{1}$$

We can see that we can trivially invert 1 in order to get world coordinates from the camera coordinates.

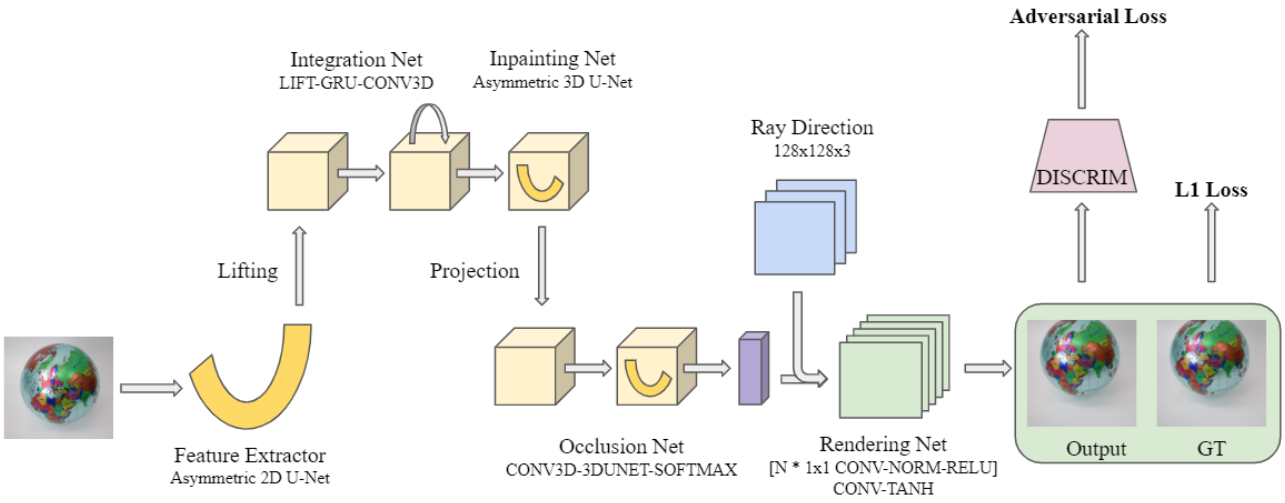

Figure 1. Overview of entire model architecture used for investigating and testing implicit and explicit specularity rendering. One should note that while the entire model is used for training, the parts of the model prior to projection become obsolete during testing.

The integration network uses a gated recurrent network (GRU), which has the advantage of avoiding vanishing/exploding gradients as in a standard RNN, while also being able to learn time-invariance (i.e. be affected by information that was "integrated" much earlier). We denote the lifted 3D volume for time t as x_t and trainable parameter weights W, hidden weights U, and biases B. We then have a somewhat standard GRU [1] with the following equations for the update gate z_t, reset gate r_t, the current "memory" s_t, and the new hidden state h_t.

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1} + B^{(z)}) \quad (2)$$

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1} + B^{(r)}) \quad (3)$$

$$s_t = max(0, W^{(s)}x_t + r_t \odot U^{(s)}h_{t-1}) \quad (4)$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t \quad (5)$$

3.2.3 Inpainting Network

The learned 3D feature representation may not have sufficient training samples to create a complete surface render, so we account for any of these gaps using a 3D U-Net (3D CONV layers instead of 2D). In this case, we have two downsampling and upsampling layers.

3.2.4 Occlusion Network

After projection via 1 back to 2D, an occlusion network is used to predict visibility of pixels in the ray "behind" each pixel in a view. After compression of these "columns" to 4-dimensional vectors, the occlusion network applies a CONV3D-3DUNET-SOFTMAX module to predict visibility of these pixels along the ray. The output from this network is essentially a depth map. We later analyze this output to gain qualitative insight into the different strategies employed by the model to embed specular effects.

3.2.5 Rendering Network

Finally, we have a stripped-down rendering network that uses 4 1x1 CONV-NORM-RELU layers followed by an additional 1x1 CONV layer and a TANH squashing function. Previous iterations of the model used 3x3 2D CONV filters to render target views. Figure 2 depicts the change in pixel value as the camera (blue) rotates around the center of the grid (e.g. the two 3x3 grids depict antipodal points in the camera's orbit about the center of the grid [both grids have the same world coordinates]). One can see that in the 3x3 case, the network can simply "look" at the surrounding pixels to infer the view direction, as the specularities (i.e. change in pixel value depending on camera pose) are view-dependent. Thus, the network can learn to "work backwards" from the surrounding specularities to infer view direction and appropriately model the specularity at the center pixel in question. However, in the 1x1 case, one can see that this context is no longer available. Thus, it is theoretically impossible for the rendering network to infer camera pose and thus specularities in the 1x1 case.

This rendering network architecture is motivated by the desire to better understand exactly where specularities may be implicitly modeled in the DeepVoxels network. By pinpointing this area, we aim to inject the explicit data necessary for DeepVoxels to more explicitly and concretely model specularities. Thus, after stripping down this net-

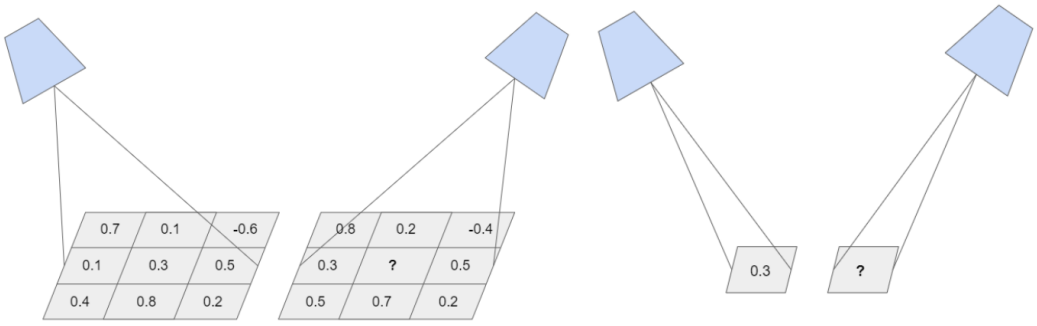

Figure 2. Comparison between rendering network's availability of data (and hence ability to infer ray direction) in 3x3 CONV case (left) and 1x1 CONV case (right).

work, we attempt to explicitly model specularities by manually computing and appending ray direction to the input feature set, as shown in Figure 2. The normalized ray direction is calculated by first performing a perspective projection according to the following equation, where (x, y, f) represent the apparent coordinates and depth from the perspective of the camera and (X, Y, Z) represent the coordinates of a point from the camera coordinates (camera at origin). The below expression [16] can be derived by noticing that the two base triangles are similar. Thus, if we have a point in the "real world" (x', y') with origin (c_x, c_y), we have the following.

$$X = \frac{(x' - c_x)Z}{F}, Y = \frac{(y' - c_y)Z}{F} \tag{6}$$

In practice, we use homogeneous coordinates to facilitate perspective projection via matrices. By convention we have the following representation of a 2-dimensional point (x', y') in some 3-dimensional space (x_H, y_H, z_H) such that

$$x' = \frac{x_H}{z_H}, y' = \frac{y_H}{z_H} \tag{7}$$

so it must be true that when we extend this system to camera world coordinates, we have the following:

$$\begin{bmatrix} x_H \\ y_H \\ z_H \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{8}$$

Thus, in practice we use the vector with the appended scalar value of 1. We can now harness the power of homogeneous coordinates and get the world coordinates from the camera world coordinates by simply multiplying by a unique 4x4 matrix representing the camera pose, which allows for translation to the appropriate coordinate system. We can then calculate the normalized ray direction by normalizing the vector difference between each pixel's world coordinates and the camera position.

$$\hat{r} = \frac{P_w - O_c}{\|P_w - O_c\|} \tag{9}$$

3.2.6 Loss

We add a weighted (in practice, optimally λ $100 - 200$) $L1$ loss to our generator loss to get the equations [5] as shown, where G and D are our generator and discriminators respectively, and y is composed of ground truth images. While an isolated $L1$ loss would not make sense as it would largely miss out on the ability to have reasonable spatial reasoning (e.g. shifting every pixel one pixel to the right would likely be catastrophic), an isolated adversarial loss also is found in practice to be less optimal than a weighted sum of the two loss metrics.

$$L_{GAN}(G, D) = \mathbb{E}_y[\log D(y)] + \mathbb{E}_z[\log(1 - D(G(z)))] \tag{10}$$

$$L_1(G) = \mathbb{E}_{y,z}[\|y - G(z)\|] \tag{11}$$

$$\min_G \max_D L(D, G) = L_{GAN}(G, D) + \lambda L_1(G) \tag{12}$$

4. Dataset

We evaluate the performance of our model on both publicly available three-dimensional scans that [10] uses as well as images taken "in the wild" of a real world object with specularities from a variety of angles. The synthetic datasets allow for tight control of both the training and test sets, so we have approximately 500 128x128 images and camera poses uniformly distirbuted over the northern half of the object (0 to +90 degrees). The test set, which consists of target camera poses and ground truth images consists of roughly 1000 viewpoints evenly distributed over an Archimedean spiral in order to test a wide range of view angles and distances. In the case of the real world capture, we do not have the capacity to exert precise, fine control over

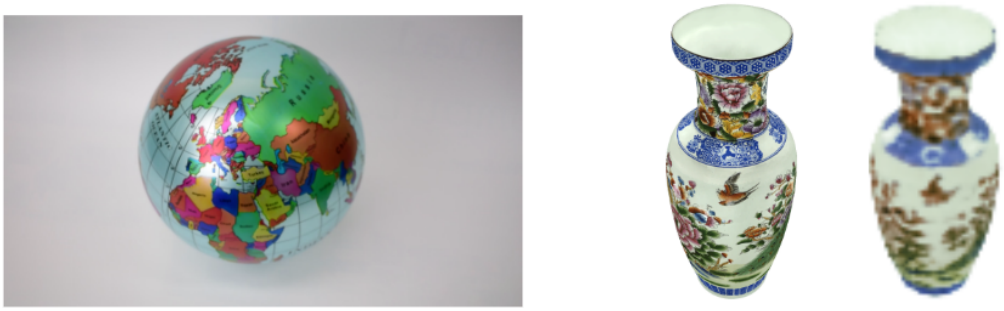

Figure 3. Left: Example of real world captured image of globe. In particular, note the prominent specular effects in the center of the globe (western Europe). Right: Example of image from synthetic dataset (center right) and generated output from "Ray" model (far right). Note the ideal centering and lack of background in this controlled render environment.

the distribution of training and testing images. After cropping our real world captured images to the desired dimensions (128x128), we have roughly 400 and 100 128x128 images in the training and test set respectively from a variety of viewing angles along with camera poses. One can watch the "video" in the code repository for this project of the real world captures to more clearly understand the composition of the dataset. As previously mentioned, we extract 32 features using our 2D U-Net in the feature extraction network, which is then fed into the rest of the model via lifting, integration, etc.

5. Results

The results of our model experiments are somewhat unexpected, but effective in providing insight into optimizing specularity rendering from 3D feature representations.

Our model is trained via a standard ADAM optimizer with a default learning rate 0.0004 chosen as it seems to converge reasonably quickly and effectively for all models. Our batch size is necessarily 1 due to the nature of the model architecture (1 view per step). The relative weights of the GAN loss and the $L1$ loss are chosen based on the qualitative observation that while the generative loss is obviously essential to effective generation (e.g. superior spatial reasoning as described previously), it tends to converge less readily and consistently than the L_1 loss as can be seen in Figure 6. Thus, to ensure continued gradient descent, we weight the L_1 loss relatively high (200 times), which seems to work well in practice (Figure 5).

In general, we evaluate the results of three different DeepVoxels-like models. The first model (Original) is simply the standard DeepVoxels architecture as outlined in [10]. The second model takes advantage of an overhauled rendering network motivated by the lack of proximal context for extraction of ray learning and implicit modeling of specularities. The third model uses perspective projection to explicitly calculate and integrate per-pixel camera ray directions that ultimately allow for explicit modeling of specularities due to the explicit nature of the viewing angle and direction.

To measure and compare the performance of our models, we not only inspect the output images on an aesthetic, qualitative level, but also via quantitative metrics. In particular, we evaluate the peak signal to noise ratio (PSNR) and structural similarity index (SSIM) to quantify model performance [4].

In the case of PSNR, the "signal" is ground truth images while the "noise" is the error of the generated render given a camera pose. Given the mean squared error MSE (average pixel-wise squared difference between ground truth and generated image) and the maximum pixel value M (e.g. 255), we have that

$$PSNR = 20\log_{10}(M) - 10\log_{10}(MSE) \tag{13}$$

We see that higher values of PSNR generally imply higher quality generated images.

We also evaluate the SSIM metric by calculating the image mean μ, image variance σ, covariance $\sigma_{1,2}$, and the range of the pixel values (e.g. 8-bit images usually 255). We also have standard parameters $k_1 = 0.01, k_2 = 0.03$.

$$SSIM = \frac{(2\mu_1\mu_2 + (k_1L)^2)(2\sigma_{1,2} + (k_2L)^2)}{(\mu_1^2 + \mu_2^2 + (k_1L)^2)(\sigma_1^2 + \sigma_2^2 + (k_2L)^2)} \tag{14}$$

Using both of these metrics (PSNR & SSIM), we yield the following results. In the non-specular case, each value represents an average over the entire vase test set, while in the specular case, each value represents an average over the entire real capture globe test set.

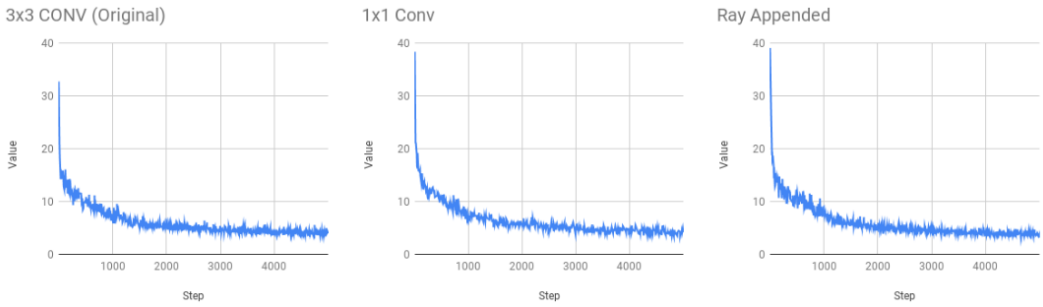

Figure 4. Loss curve for all models training on the real capture globe dataset. This is a weighted sum of generative adversarial loss and L1 loss. The loss curves for all models converge well to a reasonable loss after 5000 iterations.

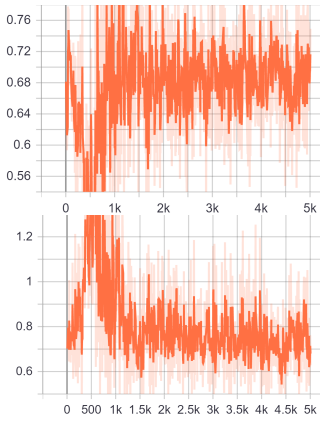

Figure 5. The difficulty of relying purely on generative loss (bottom) as a metric for progress is clearly seen in these figures of loss over iterations. The conflict between the discriminative loss (top) and the generative loss is clear here and complicates training.

Non-Specular

Metric	Original	1x1	Ray
SSIM	**0.96**	0.93	0.94
PSNR	**27.99**	25.11	26.44

Specular

Metric	Original	1x1	Ray
SSIM	0.86	0.84	**0.93**
PSNR	25.18	24.53	**28.91**

Our results shed light on the various strengths and weaknesses of the different DeepVoxels architectures tested. In the non-specular case, the original unaltered model architecture is marginally superior to both the 1x1 based architecture and the architecture with integrated per-pixel camera rays. This is not particularly surprising, especially given that the purpose of switching to 1x1 CONV nets was to limit the amount of learnable information (i.e. view angle and direction) in the network. Further, we see that providing additional feature information in the Ray model largely fails to drastically improve performance, an unsurprising result given that matte surfaces lack specularities and thus have little use for features relevant to view-dependent optics (i.e. view angle and direction).

An entirely different story is revealed in the latter set of results however on the specular globe test set. Firstly, a surprising result was that the 1x1 CONV based model continued to have comparable performance to the original DeepVoxels model despite purportedly removing the information (proximal information in 3x3 filters) necessary to be able to calculate and generate the appropriate render for view-dependent optics. It is difficult to say with certainty how this is achieved, but several hypotheses are put forward. Firstly, it is theoretically possible that a 1x1 CONV based model could learn specularities by learning a ”hole” in which a light source could shine outward. The renders would thus appear to have a reflective quality despite such ”specularities” actually just being lights shining outward. However, this theory seems unlikely given the depth maps in Figure 7. If this were indeed the case, in the 1x1 case, we could expect to see anomalies in the depth in the location where specularities purportedly appear, but this is not the case. A perhaps more promising hypothesis is that the ray direction can be extracted elsewhere in the model architecture. For example, the occlusion network may be able to combine the information along each of its rays (e.g knowing along which axes an object is thick versus thin) in order to extract camera view information. However, this requires substantial learning capacity, so it would be a surprise if this were indeed the case. We thus fail to ”break specularities” as we sought to accomplish, but nevertheless gain valuable insight into how DeepVoxels is able to effectively render view-dependent optics.

However, we still seek to understand if this implicit specularity modeling is truly effective, or if one can increase the performance of DeepVoxels through the injection of strategic information. Our results are better than anticipated, and we find that there is a rather dramatic increase in the model’s

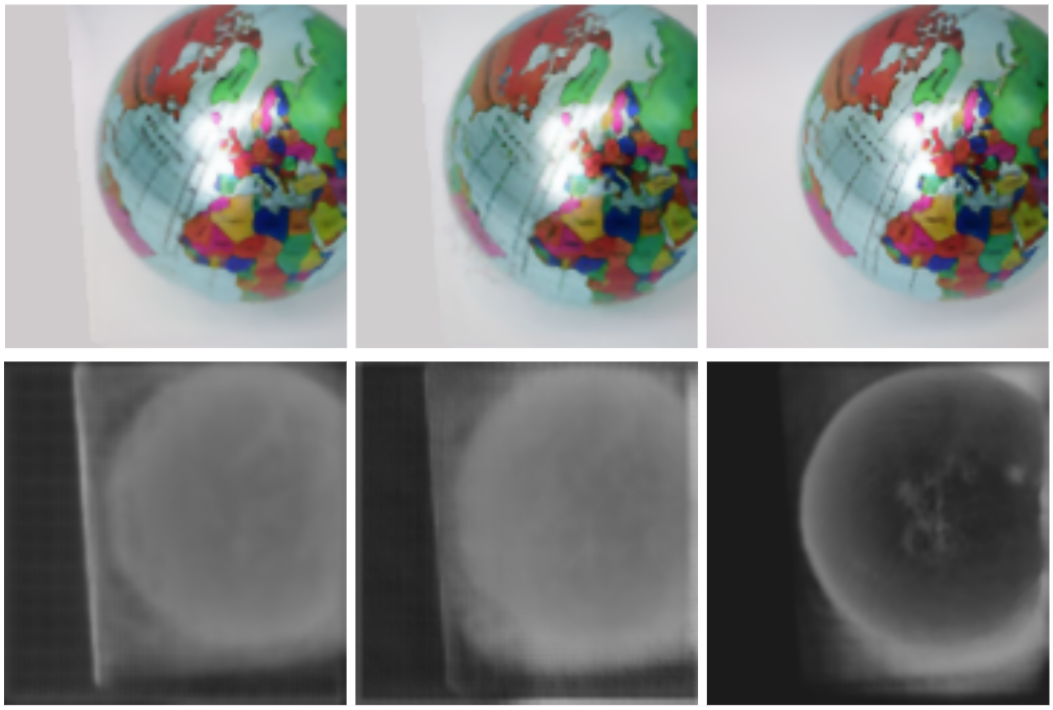

Figure 6. A randomly selected element from the test set rendered with the original, 1x1, and ray based models (from left to right). RGB output (top 3) and depth estimation (bottom 3) is depicted.

ability to accurately render specular objects when we append our generated features relevant to ray direction. It is an exciting revelation that DeepVoxels is a ”generalist” in that it can implicitly model complex features like specularities. However, our results show that DeepVoxels can be turned into a ”specialist” simply via the injection of key feature channels in strategic parts of the model architecture. The phenomenon of adding more and more layers and thus increasing memory requirements may thus be mitigated by pinpointing key leverage points at which information relevant to one's priorities may be employed.

In general, our test results seem to be fairly reliable in that they do not show signs of overfitting, as training and testing accuracies remain reasonably comparable across all experiments. Great care must be taken however to avoid oversaturating training with too many viewpoints or providing test sets too similar to viewpoints in the training set.

Finally, inspection of the RGB and depth maps of a randomly generated test sample in Figure 7 provides greater insight into the advantage of explicit specularity rendering. While the RGB outputs all appear to be more or less identical. However, the depth maps shed light on the advantages and disadvantages of the various models when modeling specularities. Firstly, one can note the outline of a cube in the baseline and 1x1 CONV case. Such an error is likely caused by specular effects in the background itself rather than the globe. In the former two cases, the model is forced to implicitly guess as to how to model the specularities in the background, and does so by assuming that the background is part of the object surface. Thus, any specularities on the globe are likely handled in a similar way. The model likely assumes that the specular effects are ”glued” to the surface of the globe. Such an assumption more or less allows for decent performance, as even though these effects are view-dependent, the effect is not so great in this case such that viewing the globe with true specularities from varying angles would result in images significantly different from a globe with a white patch appended to it. However, the performance of these models is in stark contrast to that of the model with appended camera ray features. The depth map shows little to no cuboid artifacts, meaning that the model is able to correctly learn how the background may change color due to specularities according to the camera view angle and direction. Thus, the background can remain as a discrete entity from the globe, as the model no longer tries to force the background to be part of the modeled 3D feature representation.

6. Conclusion

We have gained significant insight into the behavior of DeepVoxels regarding its strategies and limitations regarding view-dependent optical rendering. Although we fail to

fully break the network's ability to model specularities in the way that we originally expected, we are able to rule out several hypotheses regarding how DeepVoxels is actually able to model specularities. Subsequent experiments shed further light on how specularities are modeled, as we see that the original DeepVoxels architecture is vulnerable to any effects inconsistent with a matte object, as its coping mechanism is assuming that anything that seems inconsistent must be made consistent by somehow appending it to the object at hand. Our approach of feeding DeepVoxels previously implied or "hard to get" information is successful in encouraging DeepVoxels to be lazier in terms of "doing its job" of enforcing viewpoint-invariance in all regards. We thus achieve significant improvements in the performance of the DeepVoxels network on specular datasets.

Going forward, it would be desirable to continue to stress test the DeepVoxels network with complex optical phenomena (e.g. transparency, refraction, shadows). Before doing so, our experiments have shown that it is useful to isolate modules responsible for modeling, either explicitly or implicitly, different parts of the 3D feature embedding. Once these modules are isolated, one can "feed" the model information that could provide enhanced, more explicit insight to the model during training and testing.

7. Acknowledgements

Benjamin Choi was the primary contributor to this project. The initial code base [9] was taken from the original DeepVoxels repository (github.com/vsitzmann/deepvoxels). Vincent Sitzmann not only wrote the original paper and sample code off of which this entire project was based, but also directed us to datasets and gave the project direction. He is one of the kindest, most compassionate people I have ever met, and this project would not have been possible without his guidance and help along the way.

References

[1] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[2] V. Deschaintre, M. Aittala, F. Durand, G. Drettakis, and A. Bousseau. Single-image svbrdf capture with a rendering-aware deep network. *ACM Trans. Graph.*, 37(4):128:1–128:15, July 2018.

[3] S. Georgoulis, K. Rematas, T. Ritschel, E. Gavves, M. Fritz, L. Van Gool, and T. Tuytelaars. Reflectance and natural illumination from single-material specular objects using deep learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(8):1932–1947, Aug 2018.

[4] A. Hor and D. Ziou. Image quality metrics: Psnr vs. ssim. pages 2366–2369, 08 2010.

[5] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[6] C. Lin, C. Kong, and S. Lucey. Learning efficient point cloud generation for dense 3d object reconstruction. *CoRR*, abs/1706.07036, 2017.

[7] K. Regmi and A. Borji. Cross-view image synthesis using conditional gans. *CoRR*, abs/1803.03396, 2018.

[8] D. J. Rezende, S. M. A. Eslami, S. Mohamed, P. Battaglia, M. Jaderberg, and N. Heess. Unsupervised learning of 3d structure from images. *CoRR*, abs/1607.00662, 2016.

[9] V. Sitzmann. Deepvoxels. `https://github.com/vsitzmann/deepvoxels`, 2019.

[10] V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein, and M. Zollhöfer. Deepvoxels: Learning persistent 3d feature embeddings, 2019.

[11] S.-H. Sun, M. Huh, Y.-H. Liao, N. Zhang, and J. J. Lim. Multi-view to novel view: Synthesizing novel views with self-learned confidence. In *European Conference on Computer Vision*, 2018.

[12] J. Thies, M. Zollhöfer, C. Theobalt, M. Stamminger, and M. Nießner. IGNOR: image-guided neural object rendering. *CoRR*, abs/1811.10720, 2018.

[13] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y. Jiang. Pixel2mesh: Generating 3d mesh models from single RGB images. *CoRR*, abs/1804.01654, 2018.

[14] C.-H. Wei, C.-K. Chiang, Y.-W. Sun, M.-H. Lin, and S.-H. Lai. Novel multi-view synthesis from a stereo image pair for 3d display on mobile phone. In J.-I. Park and J. Kim, editors, *Computer Vision - ACCV 2012 Workshops*, pages 568–579, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[15] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1696–1704. Curran Associates, Inc., 2016.

[16] Z. Zhang. Camera calibration. *Computer vision: a reference guide*, pages 76–77, 2014.