QuickRender: Supplemental Information

Anonymous submission

QuickRender Dataset Details

A folder with a temporal image triplet and accompanying auxiliary data is included with this supplemental file. The full dataset will be made available upon publication.

Auxiliary data

Figure 1 shows an example of what each of the auxiliary data channels look like for a single image of the dataset (column 1). In order from left to right beginning with column 2, the auxiliary data channels are as follows:

Depth: Distance from the camera to the object, with high values being close and low values being far. Presented as a grayscale image.

Normals: The normal vector for the surface of each object. Presented as an RGB image with the R G and B channels representing the X, Y and Z euler angles of the normal vector respectively. All normals are regularized to be relative to the camera's angle.

Object Segmentation: Presented as a grayscale image with different values for each object. Each object receives a unique value with each seed.

Velocity: 3D velocity of the object surface along the camera plane. Represented by an RGB image. R and G channels represent the X and Y planar velocities, respectively. B represents motion toward or away from the camera. Images are ranged [0,1], re-ranging to [-1,1] makes velocities represent [-100,100] pixel movement along the camera plane. Positive magnitudes of the RGB channels after this re-ranging indicate rightward, downward, and toward the camera motion respectively.

Procedural Scene Generation

Figure 2 shows the diversity that is produced by a single scene within the QuickRender dataset. The boundaries of the procedural variables are tightly controlled to ensure an extremely low level of data cleaning is required after a set of images are generated. Each scene uses a unique set of methods depending on the scene's requirements. A few common methods that are used are as follows:

Beziers: Cameras and moving objects in a scene follow a fixed bezier path. Their position on this path and speed moving along it are randomized with each seed generation. Also, if applicable, a random offset vector for their position from the bezier is generated. This helps ensure natural object and camera motion, as well as preventing impossible object or camera positions, such as clipping or extreme camera occlusion.

Shaders: Each surface shader for an object is procedural, allowing their properties to be heavily randomized with each random seed. This includes, but is not limited to: surface color, roughness, metallicity, surface normals, and light emission. Additionally, since each shader is procedural, all noise textures used to generate the shader are randomized. This is done by using the random seed as the ω component of the 4D input vector for the noise textures.

Lighting: Lighting is randomized by time of day and weather. A high dynamic range image (HDRI) is selected from a pool for the given time of day and weather. This is then augmented by randomizing rotation, brightness and saturation. In cases where the time of day or weather affects aspects of the scene other than lighting, these are randomized too. For example, adding and randomizing puddles on flat surfaces in rainy weather, and turning on lights and emissive textures during dark scenes. It is worth noting that some scenes physically simulate lighting rather than using HDRIs. For these, many more parameters of the lighting are randomized, such as size and density of volumetric clouds, atmospheric diffraction, as well as the previously mentioned parameters.

Scene Diversity

Lastly, figure 3 shows the level of diversity that is present between the 5 scenes. Each contains a unique set of objects and shaders that helps to increase diversity across the final dataset. All scenes contain hundreds of unique objects that are randomized with each procedural seed.

From left to right, scenes 1 and 4 are representative of exploded view diagrams for product rendering, while scene 2 is a nature scene containing fully rendered mountains, trees, birds, and many moving objects. Scenes 3 and 5 are driving scenes in the city and country, respectively.

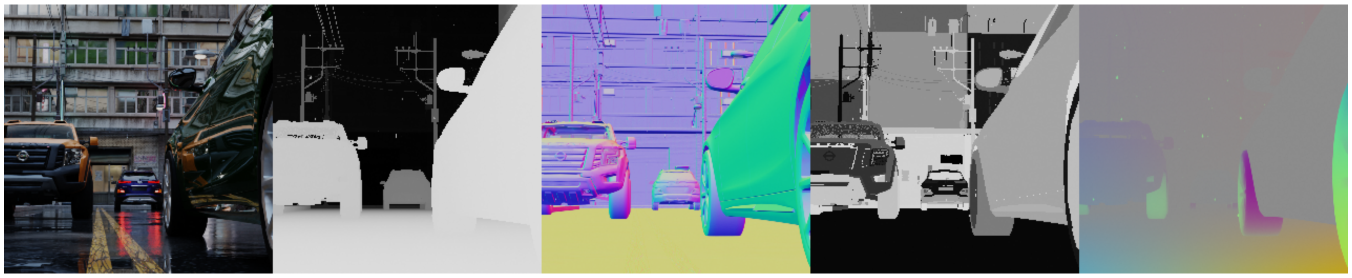
Figure 1: Sample of auxiliary data provided with each image. From left to right: visual image, depth map, surface normals, object segmentation map, velocity map

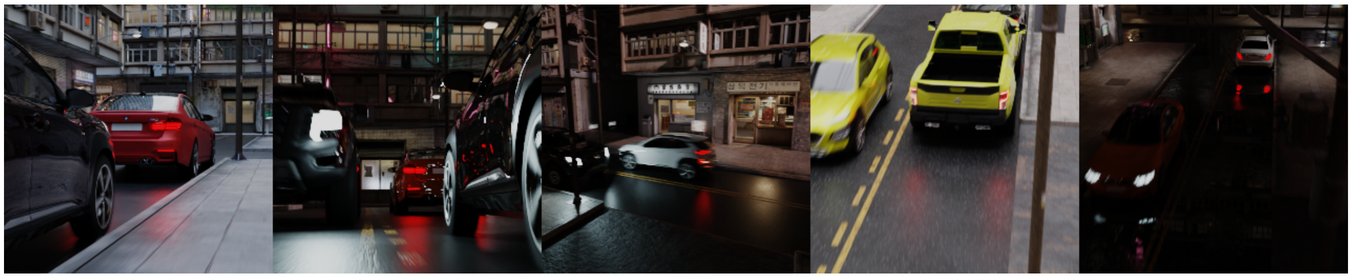
Figure 2: Sample of diversity of images produced within a single scene

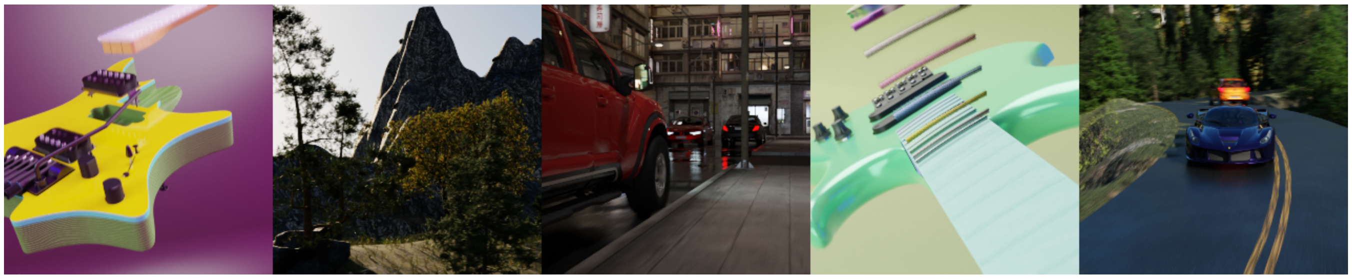
Figure 3: Sample of the 5 unique procedural scenes

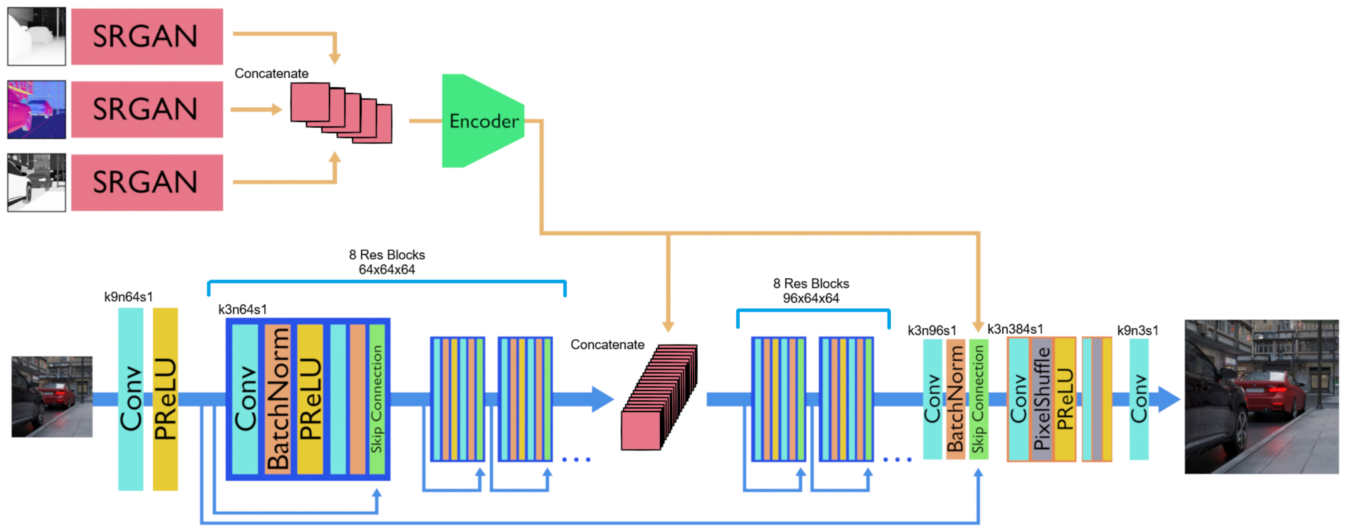

Figure 4: High level diagram of the MetaSRGAN network architecture

MetaSRGan

Figure 4 shows a diagram of the MetaSRGAN network. Auxiliary data is input at the same resolution as the input image. The auxiliary data is then upscaled by 4x with a pretrained SRGAN implementation. All channels from the normal map, and one channel from each of the depth and object maps are concatenated to form the input tensor for the autoencoder. This encodes the metadata into a 32 filter tensor with the same dimensions as the low resolution inputs, forming a metavector.

The structure does not deviate from SRGAN until the 8th residual block. Here, the current tensor is concatenated with the encoded metavector. This changes the number of filters in the remaining residual blocks, meaning the final skip connection also requires the concatenation of the auto encoded metavector before performing the element wise sum.

Upsampling is performed with a pixel shuffle algorithm as in the SRGAN paper. This was tested alongside other upsampling methods such as nearest neighbor and bilinear. Pixel shuffle was the most successful at reconstructing fine detail in the super resolved image. However, this comes at the expense of pixel level checkerboard artifacts. These are obviously present in the SRGAN implementation, however they diminish in magnitude significantly in the final version of MetaSRGAN.

SRGAN and the auxiliary data autoencoder are trained separately from the MetaSRGAN. Training these networks simultaneously introduced instability in the training process. Using the trained networks as initial weights has potential to stabilize this. However, given the hardware restrictions present, training these networks in tandem reduces the maximum batch size too significantly for it to be a viable option. For these reasons, independent training made more sense in this case.

During training, data was augmented using random horizontal and vertical flips, as well as random size crops. QuickRender has 512x512 as the full resolution. A size of 256x256 was used to train MetaSRGAN. With each sample of an image, a random size between 256x256 and 512x512 is selected. A random crop of this size is taken, then it is downsampled to 256x256 using BiCubic downsampling.

The trained versions of SRGAN and MetaSRGAN were tested on the validation set of QuickRender. Table 1 shows the SSIM and PSNR metrics for both networks, demonstrating improved performance correlating to visual assessment for MetaSRGAN. As noted in the SRGAN paper, these metrics do not necessarily indicate improvement in the domain of super resolution. Ideally, mean opinion score (MOS) would be used in follow-up research. A sample image from outside the QuickRender dataset can be seen in figure 5. This example was taken from a static blender scene rendered at 1024x1024 resolution for full resolution ground truth, and 256x256 for low resolution input. The comparative examples show that MetaSRGAN improves on SRGAN in the ability to reconstruct both colors and shapes. These results were consistent with other scenes that were tested, but were most prominent with inorganic shapes and vibrant colors.

Metric	SRGAN	MetaSRGAN
SSIM	0.64402	0.69584
PSNR	23.32308	26.41284

Table 1: Performance metrics for SRGAN and MetaSRGAN

Inference Time

As the original goal in developing the QuickRender dataset was to accelerate the rendering process for high resolution images, testing was done with the trained MetaSRGAN to assess the efficiency of upscaling a small rendered image instead of rendering the full resolution image. Blender splash screens 3_5 and 3_2 were used as test cases. Table 2 shows the results of these tests, illustrating an average time savings of 90.5%.

Image	Rendered	Inferred	Time Savings
3_2 2k	09:52:65	00:48:62	91.80%
3_2 4k	37:59:91	02:45:02	92.76%
3_5 2k	03:35:24	00:29:07	86.49%
3_5 4k	13:40:42	01:13:50	91.04%

Table 2: Time taken to produce a full resolution image (mm:ss:ms) via rendering or inferred with MetaSRGAN

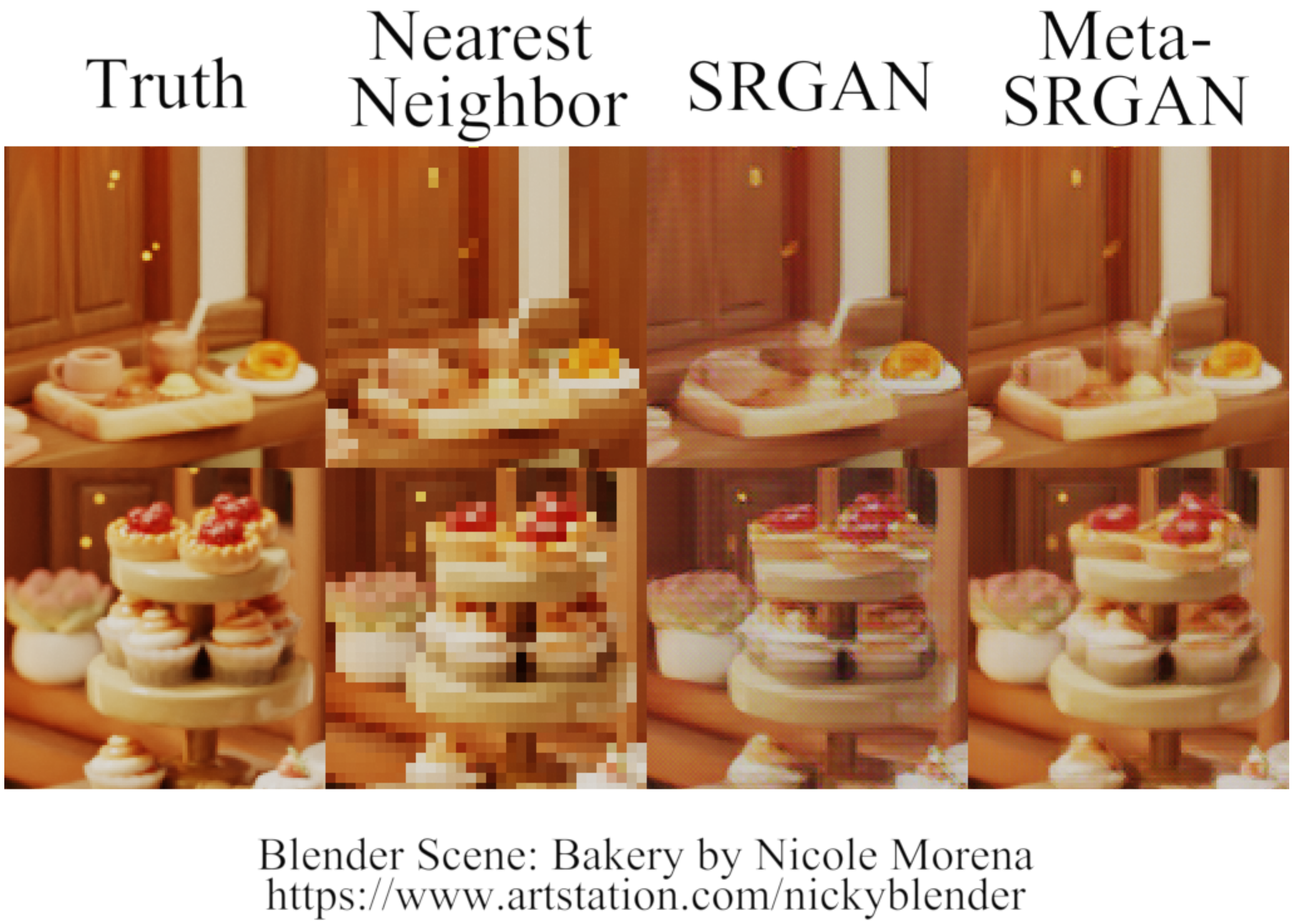
Truth
Nearest Neighbor
SRGAN
Meta-SRGAN
Blender Scene: Bakery by Nicole Morena
https://www.artstation.com/nickyblender

Figure 5: Sample image from outside the QuickRender dataset upsampled three ways