Machine Learning Assisted Rendering Techniques

Yinqi (Bill) Sun
New York University
ys3540@nyu.edu

Honor Pledge:

"I affirm that I will not give or receive any unauthorized help on this academic activity and that all the work I submitted is my own."
Machine learning homework for week 7. **Signature:** Yinqi Sun **Date:** Mar.29 2021

1. INTRODUCTION

Vision has been one of the most important ways that human uses to perceive the world because of the high expressiveness of an image and the information density it conveys. For example, a picture can represent highly complicated concepts through different colors, shapes, and positions; and people can perceive all of these with just a blink of an eye.

For computer systems to generate an image, it often goes through the rendering process, which I will briefly address in Chapter 2. However, rendering can be a demanding process, which requires a precisely defined model we called *scene* and the whole rendering process can be computationally expensive.

To solve these problems, in Chapter 3, I will review the method Generative Query Network (GQN) [1] which uses unsupervised deep neural networks that given only images observed from several viewpoints of an unknown scene can generate an observation of that scene from a viewpoint it has not seen before and without going through traditional rendering pipelines. Then in Chapter 4 I'll review some improvements [2, 3, 4] over GQN that strengthened its capabilities in specific applications.

2. RENDERING

Rendering typically starts from a *scene* that describes the properties of the things we may need to show, including colors, shapes, positions, and other physical properties. Then, the scene, along with additional parameters specifying how the scene is observed, i.e., the position of our eyes (viewpoint), is fed into the *rendering pipeline*, where the geometries of the scene will first be transformed, projected, and clipped into a 2D frame, then, every visible surface is divided into fragments and assigned to the corresponding pixel of the output image. Finally, the color of a pixel is determined by the color of the object it corresponds to and the properties of the light sources in the environment.

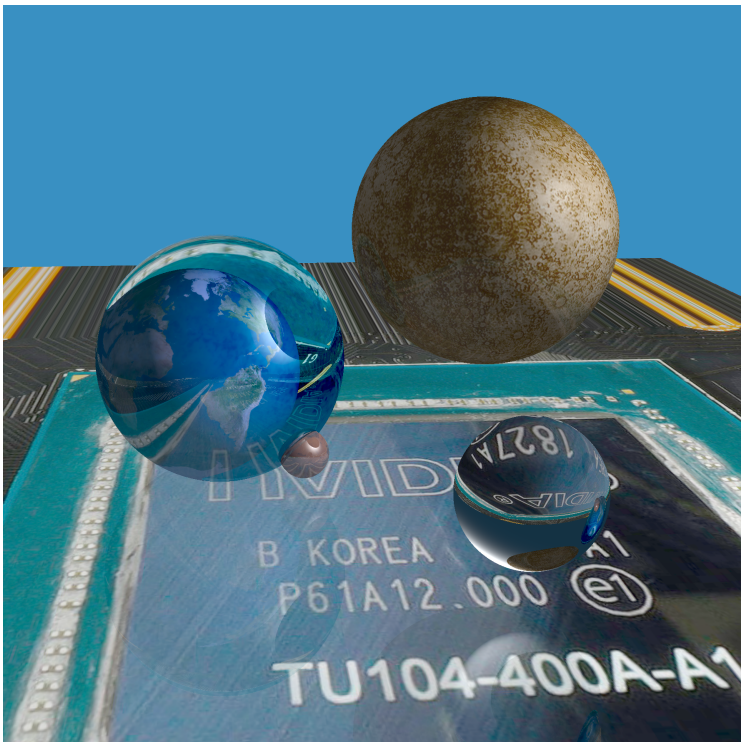


Figure 1. Realtime interactive raytracing on an extremely simple scene. [5]

As an important role in human-computer interaction (HCI), rendering has seen huge advancement in quality during the last decade. With more powerful hardware, we can now render much more complicated scenes with over a million triangles and use more realistic rendering techniques like ray tracing. However, the obstacles ahead are also obvious; first of all, we may not be able to get the model every time due to real-world restrictions. For example, if we are constructing a scene from hand drawings or street camera images. Then, rendering may not be a cheap process, even with modern hardware, high fidelity rendering, i.e., ray tracing, at interactive speed is no easy task. Because we need to reflect the user's action onto the image quick enough so that the user will not feel the delay, the whole rendering pipeline typically completes within 40ms. Figure 1 shows one of my homework that does raytracing at interactive speed via WebGL because the objects are not triangulated; there are only 6 objects instead of thousands of triangles in the scene. However, we can not easily render complex shapes without triangulation.

3. GENERATIVE QUERY NETWORK

People gave many answers to those two obstacles above. For example, there're works focusing on inference of the 3D scene from 2D images by training a neural network that constructs the depth map and contour from the images and synthesizes a 3D point cloud from them [6]. Another recent work [7] uses GAN to synthesize 3D point-cloud from 2D images. However, these representations are discrete and only sparsely sampling the underlying smooth surfaces [8], and the image rendered from the point cloud or voxels can be vastly different from the observations because of different rendering technique/parameter may be used.

Imagine when we first see a scene from an angle; it is natural for us to visualize how it would look like from other angles. This is probably because when seeing the scene, we get an 'idea' of how the scene is configured. Along with some a-priori knowledge about the world, i.e., how does the object typically looked like, we can generate an 'imagination' of the scene from an unseen angle. This is similar to how Generative Query Network (GQN) [1] works which makes it another interesting answer to the above questions. **Figure 2.** below shows how it works. The 4 observations on the top are observed from the scene at viewpoints shown as gray cameras on the map. The yellow camera represents the unseen query viewpoint and the image labeled 'neural rendering' is the estimated image from that viewpoint generated by GQN compared to the ground truth image on the left.

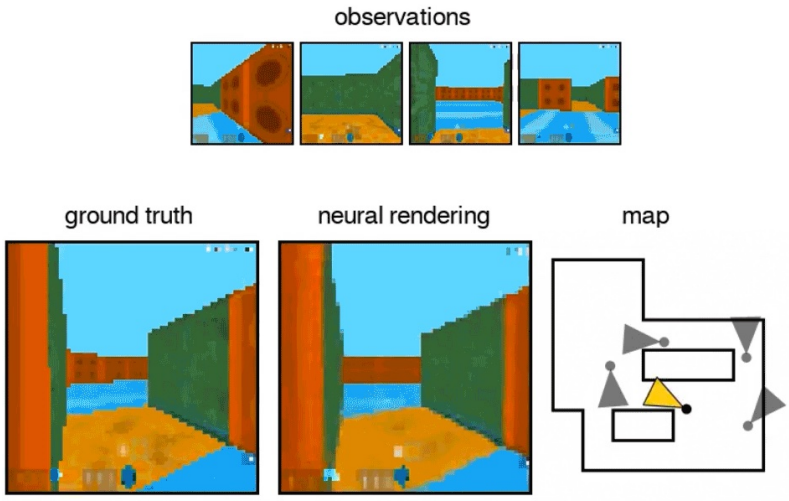


Figure 2. GQN Renders the scene based on observations from different viewpoints.

As shown in **figure 3.** below, GQN consists of a representation network **f** and a generative network **g**. The representation network gets different 2D observations of the 3D scene as input and will extract a representation vector **r** that encodes the information about the scene itself, including the colors, shapes and positions. The generative network will take the scene representation **r** and a query viewpoint $\mathbf{v}^q$ that we need it to render the scene from and output the predicted observation of the scene at $\mathbf{v}^q$ as image **x**.

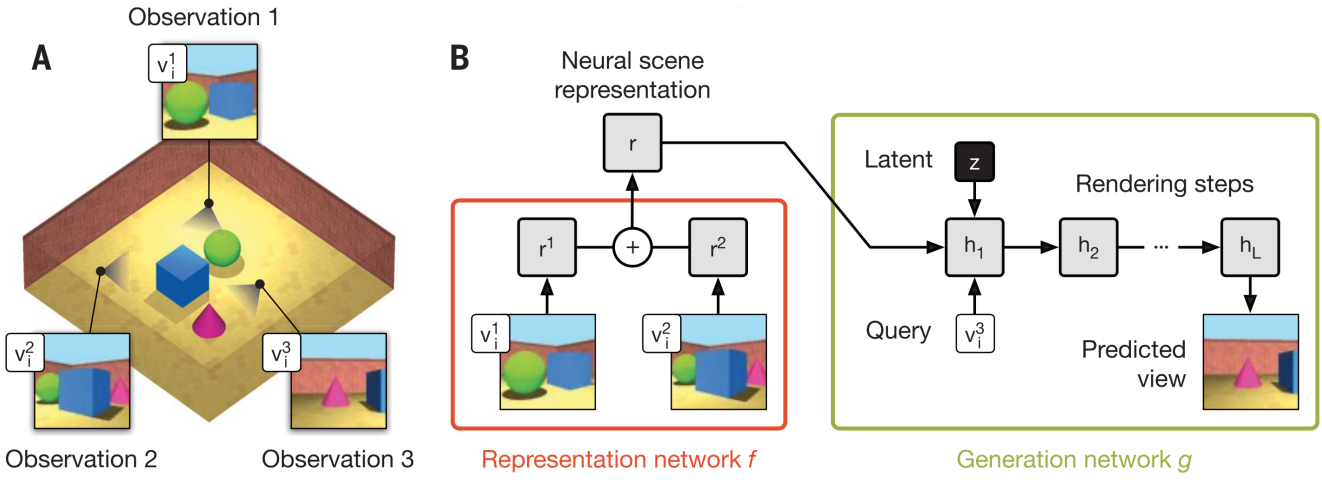


Figure 3. Schematic illustration of the Generative Query Network.

3.1 Representation Network

The Representation Network used by GQN is typically a convolutional network. It takes the viewpoint **v** and **x** the image observed from **v** as input and will output a viewpoint independent feature vector **r**. The image **x** is represented by a matrix of dimension $width \times height \times channels$. In figure 4. we have an 64×64 image with 3 channels for RGB color. The viewpoint $\mathbf{v} = (w_x, w_y, w_z, cos(y), sin(y), cos(p), sin(p))$ is a $1 \times 1 \times 7$ vector defined by 5 parameters, the camera position coordinate $\mathbf{w}=(w_x, w_y, w_z)$, camera rotation angle on y-axis (yaw) y and camera rotation on x-axis (pitch) p. For each input observation $\mathbf{o}^i = (\mathbf{x}^i, \mathbf{v}^i)$, the convolution network $\psi(\mathbf{x}^i, \mathbf{v}^i)$ will output the extracted scene representation from that image $\mathbf{r}^i$. Because all the vectors $\mathbf{r}^i$ are representing a same scene, we simply sum them up to aggregate features extracted from all the observations.

One possible architecture of the convolutional network $\psi(\mathbf{x}, \mathbf{v})$ is shown on **Figure 4.** below. Where k and s in the figure are the kernel size and stride size respectively. Black arrows represent convolutional layers followed by ReLUs and red arrows marked with '+' indicate residual connections.

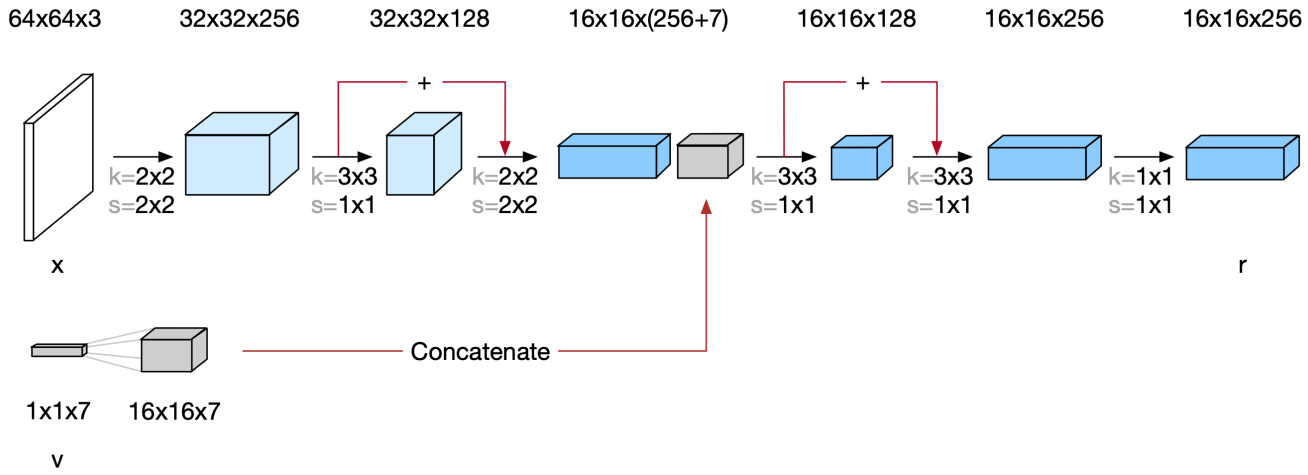


Figure 4. Representation network.

The mathematical definition of the representation network **f** is shown below:

$$\mathbf{v}_i = (\mathbf{w}_i, cos(y_i), sin(y_i), cos(p_i), sin(p_i)) \tag{1}$$

$$\mathbf{f}(\mathbf{x}_{1,\ldots,n}, \mathbf{v}_{1,\ldots,n}) = \sum_{i=1}^{n} \psi(\mathbf{x}_i, \mathbf{v}_i) \tag{2}$$

3.2 Generative Network

(See Ch. 1.4 from the Supplementary Material of this paper for original descriptions.)

Basically, what we are left to do is to generate the image from the scene representation **r**, the query viewpoint $\mathbf{v}^q$ and maybe some information θ that are learned from training with queries and its ground truth from different scene. We can fit this process with conditional latent variable models where the resulting image **x** can be drawn from the distribution density function $g_\theta(\boldsymbol{x}|\boldsymbol{v}^q, \boldsymbol{r})$. If we introduce latent variables **z**, which correspond to features that are not learned from the training set but associated with input (in this case, **r**, $\mathbf{v}^q$), we would have,

$$g_\theta(\boldsymbol{x}|\boldsymbol{v}^q, \boldsymbol{r}) = \int g_\theta(\boldsymbol{x}|\boldsymbol{v}^q, \boldsymbol{r}, \boldsymbol{z}) \pi_\theta(\boldsymbol{z}|\boldsymbol{v}^q, \boldsymbol{r}) d\boldsymbol{z} \tag{3}$$

where π_θ here is the conditional prior probability indicating the probability density of latent variable **z** when $\boldsymbol{v}^q, \boldsymbol{r}$ happens. θ here is parameters that can be learned by training.

This idea is very similar to another paper [9] from DeepMind. We first divide the latent variables **z** into L groups, and by constructing the density function $\pi_\theta(\boldsymbol{z}|\boldsymbol{v}^q, \boldsymbol{r})$ sequentially, we can calculate it as an autoregressive density:

$$\pi_\theta(\mathbf{z}|\boldsymbol{v}^q, \boldsymbol{r}) = \prod_{l=1}^{L} \pi_{\theta_l}(\mathbf{z}_l|\boldsymbol{v}^q, \boldsymbol{r}, \boldsymbol{z}_{<l}) \tag{4}$$

As shown in **Figure 5. A**, for each step, we used a size-preserving convolutional LSTM network C_θ to extract salient information from the input $\mathbf{v}^q$, **r** and latent vector $\mathbf{z}_l$ as hidden state h_l^g where $\mathbf{z}_l$ is approximated from previous iteration l. h_l^g is used to parameterize the distribution of the next latent variable $\mathbf{z}_{l+1} \sim \pi_{\theta_l}(\cdot|\boldsymbol{v}^q, \boldsymbol{r}, \boldsymbol{z}_{<l+1})$ as well as finally generating the distribution

of $\mathbf{x}$. To estimate π_θ from h_l^g, a convolutional network η_θ^π is used to parameterize (generate the mean and variance of) a Gaussian distribution that approximates π_θ from h_l^g. Because the output h_l^g from each layer is somehow correlated to $g_\theta(\cdot|\boldsymbol{v}^q, \boldsymbol{r}, \boldsymbol{z}_l)\pi_\theta(\boldsymbol{z}_l|\boldsymbol{v}^q, \boldsymbol{r}, \boldsymbol{z}_{<l})\boldsymbol{z}_l$, we cumulated the transposed convolution of each h_l^g as u_L. And because the rhs of (3) has the discrete form of $\sum_{l=1}^{L} g_\theta(\boldsymbol{x}|\boldsymbol{v}^q, \boldsymbol{r}, \boldsymbol{z}_l)\pi_\theta(\boldsymbol{z}_l|\boldsymbol{v}^q, \boldsymbol{r}, \boldsymbol{z}_{<l})\boldsymbol{z}_l \approx u_L$ we can use another convolutional network η_θ^g to map u_L to the mean value μ of $g_\theta(\boldsymbol{x}|\boldsymbol{v}^q, \boldsymbol{r})$. Using another Gaussian distribution with mean μ and variance μ_t is the pixel variance we can draw an output image $\mathbf{x}$. The left of **Figure 5.** clipped from the supplementary material contains symbolic definition of the process described above.

The reason we choose LSTM network to extract h_l is because its proven track record for handling long-range dependencies in real sequential data [9] but in theory it can be any other RNN.

$$\text{Scene encoder} \qquad \mathbf{r} = f\left(\mathbf{x}^{1,\ldots,M}, \mathbf{v}^{1,\ldots,M}\right) \tag{S9}$$

$$\text{Initial state} \qquad (\mathbf{c}_0^g, \mathbf{h}_0^g, \mathbf{u}_0) = (\mathbf{0}, \mathbf{0}, \mathbf{0}) \tag{S10}$$

$$\text{Prior factor} \qquad \pi_{\theta_l}(\cdot|\mathbf{v}^q, \mathbf{r}, \mathbf{z}_{<l}) = \mathcal{N}\left(\cdot|\eta_\theta^\pi(\mathbf{h}_l^g)\right) \tag{S11}$$

$$\text{Prior sample} \qquad \mathbf{z}_l \sim \pi_{\theta_l}(\cdot|\mathbf{v}^q, \mathbf{r}, \mathbf{z}_{<l}) \tag{S12}$$

$$\text{State update} \qquad \left(\mathbf{c}_{l+1}^g, \mathbf{h}_{l+1}^g, \mathbf{u}_{l+1}\right) = C_\theta^g\left(\mathbf{v}^q, \mathbf{r}, \mathbf{c}_l^g, \mathbf{h}_l^g, \mathbf{u}_l, \mathbf{z}_l\right) \tag{S13}$$

$$\text{Observation sample} \qquad \mathbf{x} \sim \mathcal{N}\left(\mathbf{x}^q|\mu = \eta_\theta^g(\mathbf{u}_L), \sigma = \sigma_t\right), \tag{S14}$$

where the convolutional networks $\eta_\theta^\pi(\mathbf{h}_l^g)$ map its respective inputs to the sufficient statistics of a Gaussian density (i.e., means and standard deviations) and $\eta_\theta^g(\mathbf{u}_L)$ maps its inputs to the mean of Gaussian density, and the bulk of the computation at every layer is performed by the core C_θ^g, which is a skip-connection convolutional LSTM network defined by the equations

$$\text{Convolutional LSTM state update} \qquad \left(\mathbf{c}_{l+1}^g, \mathbf{h}_{l+1}^g\right) = \text{ConvLSTM}_\theta^g\left(\mathbf{v}^q, \mathbf{r}, \mathbf{c}_l^g, \mathbf{h}_l^g, \mathbf{z}_l\right) \tag{S15}$$

$$\text{Skip connection state update} \qquad \mathbf{u}_{l+1} = \mathbf{u}_l + \Delta\left(\mathbf{h}_{l+1}^g\right), \tag{S16}$$

and $\mathbf{c}_l^g$ and $\mathbf{h}_l^g$ are the standard LSTM state variables (output and cell), ConvLSTM_θ^g is a size-preserving convolutional LSTM network and $\Delta\left(\mathbf{h}_{l+1}^g\right)$ is a transposed convolution which has the effect of up-sampling the image. Note that we use spatial $\mathbf{c}_l^g$ and $\mathbf{h}_l^g$ variables, to take advantage of the natural structure of images, and empirically we find this to outperform a fully-connected architecture. For all variables, the superscript g indicates that the corresponding variable is specific to the generative process, as opposed to the superscript e which will indicate below that the variable belongs to the encoder network in the inference process.

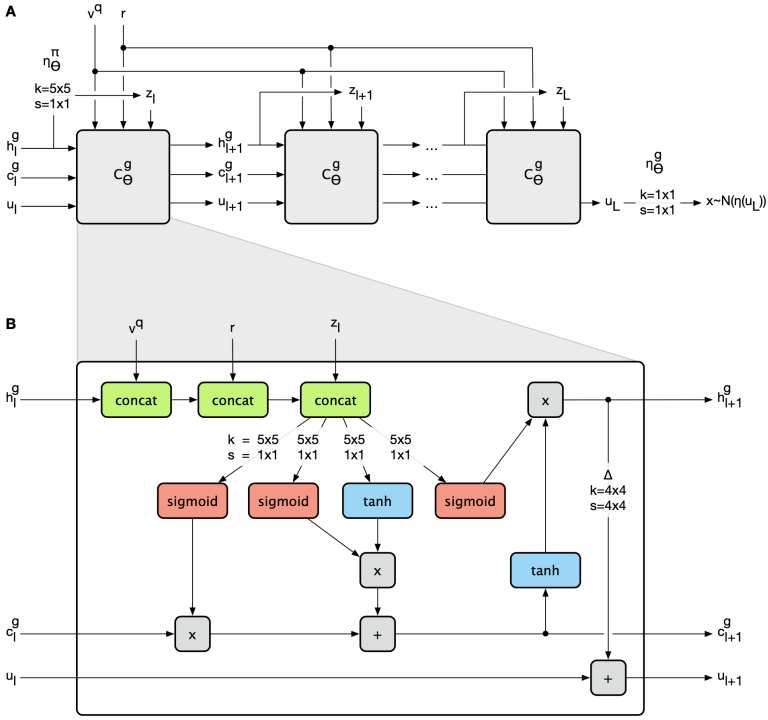


Figure 5. Generative network and the interior of a LSTM Cell (B).

3.3 Training Process of Generation Network

Before we can use the generative network g, we'd better train it with observations, queries and ground truth from each query. The intuition is that we want a θ that maximizes all the probabilities of generating the ground truth when given the observations $\mathbf{o}_i$, which is encoded into scene representation $\mathbf{r}_i$, and query viewpoint $\mathbf{v}_q$ that is $g_\theta(\mathbf{x}_i|\mathbf{v}_i^p, \mathbf{r}_i)$. And in order to focus more on data points that we're not doing so well, that is the probability g_θ^i is close to 0, instead of maximizing the sum of g_θ^i, we minimize $\mathcal{L}(\theta) = -\sum ln\ g_\theta(\mathbf{x}_i|\mathbf{v}_i^p, \mathbf{r}_i)$. $\mathcal{L}(\theta)$ here is called the negative log-likelihood. This optimization isn't easy to solve directly because according to eq, (3) it require integral over the high-dimensional latent variables z. Therefore, we minimize a lower-bound $\mathcal{F}$ (evidence lower bound, ELBO) of $\mathcal{L}(\theta)$ defined by:

$$\mathcal{F}(\theta, \phi) = \sum_i \int q_\phi(z_i|x_i, y_i) ln\ \frac{q_\phi(z_i|x_i, y_i)}{g_\theta(x_i|z_i, y_i)\pi_\theta(z_i|y_i)} dz_i = -\mathcal{L}(\theta) + \sum_i KL[q_\phi(\cdot|x_i, y_i)||p_\theta(\cdot|x_i, y_i)] \geq -\mathcal{L}(\theta) \tag{5}$$

where $y_i = \{v_i^p, r_i\}$. $q_\phi(z|x, y)$ is the approximated (variational) posterior density and p_θ is the exact posterior. In the training process, we use the same way to parameterize $q_\phi(z|x^q, v^q, r)$ as we did with the prior distribution of π_θ in the generator. (See Ch. 1.5 of the Supplementary Material). The real posterior of $\mathbf{z}_l$ is given by the distribution we used to draw $\mathbf{z}_l$ in the generator which is the prior π_{θ_l}. And the likelihood $\mathcal{L}(\theta)$ here is acquired by the probability of drawing the ground truth $\mathbf{x}^q$ from the distribution of $\mathbf{x}$ output by the generator (see Ch. 3.2), which is a Gaussian distribution parameterized by u_L and σ_t. The ELBO $-\mathcal{F}(\theta, \phi)$ is then calculated as shown in eq. (5) by adding the log-likelihood $\mathcal{L}(\theta) = ln(\mathcal{N}(\boldsymbol{x}^q|\mu = \eta_\theta^g(\boldsymbol{u}_L, \sigma = \sigma_t)))$ the sum of negative KL distances of $q_{\phi_l}(\cdot, x^q, v^q, r, z_{<l})$ and $\pi_\theta(\cdot, v^q, r, z_{<l})$ for each latent variable $\boldsymbol{z}_l$. This process is shown in Algorithm S2 in **Figure 5.**

The above ELBO is then backpropergated using the re-parameterization trick [10] to get the gradient of ELBO with respect to parameters θ and ϕ. The gradients $\nabla_\theta ELBO, \nabla_\phi ELBO$ along with a learning rate μ_t are then used to update θ and ϕ using Adam's algorithm (an adaptive gradient descent method). This training process is repeated for S_max iterations where in each iteration, a mini-batch of B scene each with M observations and a query is drawn from the dataset. The learning rate is annealed so that it learns quickly at first and stabilizes overtime. This process is described in Algorithm S1 in **Figure 5.**

Algorithm S1: GQN training loop.

Data: Choose dataset D from Room, Jaco, Labyrinth or Shepard-Metzler
Input: Initial parameters θ and ϕ. Optimizer parameters μ_i, μ_f, n, S_{max}, σ_i, σ_f, β_1 and β_2
Output: Learned parameters θ and ϕ

def *SampleBatch*(B, M, K)**:**
  /* Sample number of views */
  $M \sim \text{Uniform}(0, K)$
  /* Initialize data batch */
  $D = \{\}$
  for $b \leftarrow 0$ **in** $(B-1)$**:**
    /* Sample scene index */
    $i \sim \text{Uniform}(0, N-1)$
    for $k \leftarrow 0$ **in** $(M-1)$**:**
      /* Sample view */
      $(\mathbf{x}_i^k, \mathbf{v}_i^k) \sim$ scene i
      $D \leftarrow D + \{(\mathbf{x}_i^k, \mathbf{v}_i^k)\}$
    /* Sample query view */
    $(\mathbf{x}_i^q, \mathbf{v}_i^q) \sim$ scene i
    $D \leftarrow D + \{(\mathbf{x}_i^q, \mathbf{v}_i^q)\}$
/* Training Iterations */
for $t \leftarrow 0$ **in** $(S_{max}-1)$**:**
  $D \leftarrow \text{SampleBatch}(B, M, K)$
  ELBO $\leftarrow$ EstimateELBO(D, σ_t) (Algorithm S2)
  /* Compute empirical ELBO gradients */
  ∇_θELBO, ∇_ϕELBO $\leftarrow$ Backprop(ELBO).
  /* Update parameters */
  $\theta, \phi \leftarrow$ Optimizer(∇_θELBO, ∇_ϕELBO, μ_t)
  /* Update optimizer state */
  $\mu_t \leftarrow \max\left(\mu_f + (\mu_i - \mu_f)\left(1 - \frac{t}{n}\right), \mu_f\right)$
  /* Pixel-variance annealing */
  $\sigma_t \leftarrow \max\left(\sigma_f + (\sigma_i - \sigma_f)\left(1 - \frac{t}{n}\right), \sigma_f\right)$

Algorithm S2: Generating a sample from the approximate variational GQN posterior and estimating the ELBO.

Input: Observed views $\{(\mathbf{x}^k, \mathbf{v}^k)\}$, query camera: $\mathbf{v}^q$, target image: $\mathbf{x}^q$, pixel-variance: σ_t
Output: Sample from the posterior $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}^q, \mathbf{v}^q, \mathbf{r})$, empirical estimate of the ELBO

def *EstimateELBO*($\{(\mathbf{x}^k, \mathbf{v}^k)\}, (\mathbf{v}^q, \mathbf{x}^q), \sigma_t$)**:**
  Output: Empirical estimate of the ELBO
  /* Scene encoder */
  $\mathbf{r} \leftarrow 0$
  for $k \leftarrow 0$ **in** $(M-1)$**:**
    $\hat{\mathbf{v}}^k \leftarrow (\mathbf{w}^k, \cos(\mathbf{y}^k), \sin(\mathbf{y}^k), \cos(\mathbf{p}^k), \sin(\mathbf{p}^k))$
    $\mathbf{r}^k \leftarrow \psi\left(\mathbf{x}^k, \hat{\mathbf{v}}^k\right)$
    $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{r}^k$
  /* Generator initial state */
  $(\mathbf{c}_0^g, \mathbf{h}_0^g, \mathbf{u}_0) \leftarrow (\mathbf{0}, \mathbf{0}, \mathbf{0})$
  /* Inference initial state */
  $(\mathbf{c}_0^e, \mathbf{h}_0^e) \leftarrow (\mathbf{0}, \mathbf{0})$
  ELBO $\leftarrow 0$
  for $l \leftarrow 0$ **in** $(L-1)$**:**
    /* Prior factor */
    $\pi_{\theta_l}(\cdot|\mathbf{v}^q, \mathbf{r}, \mathbf{z}_{<l}) \leftarrow \mathcal{N}\left(\cdot|\eta_\theta^\pi(\mathbf{h}_l^g)\right)$
    /* Inference state update */
    $\left(\mathbf{c}_{l+1}^e, \mathbf{h}_{l+1}^e\right) \leftarrow C_\phi^e(\mathbf{x}^q, \mathbf{v}^q, \mathbf{r}, \mathbf{c}_l^e, \mathbf{h}_l^e, \mathbf{h}_l^g, \mathbf{u}_l)$
    /* Posterior factor */
    $q_{\phi_l}(\cdot|\mathbf{x}^q, \mathbf{v}^q, \mathbf{r}, \mathbf{z}_{<l}) \leftarrow \mathcal{N}\left(\cdot|\eta_\theta^e(\mathbf{h}_l^e)\right)$
    /* Posterior sample */
    $\mathbf{z}_l \sim q_{\phi_l}(\cdot|\mathbf{x}^q, \mathbf{v}^q, \mathbf{r}, \mathbf{z}_{<l})$
    /* Generator state update */
    $\left(\mathbf{c}_{l+1}^g, \mathbf{h}_{l+1}^g, \mathbf{u}_{l+1}\right) \leftarrow C_\theta^g(\mathbf{v}^q, \mathbf{r}, \mathbf{c}_l^g, \mathbf{h}_l^g, \mathbf{u}_l)$
    /* ELBO KL contribution update */
    ELBO $\leftarrow$ ELBO $-$ KL$\left[q_{\phi_l}(\cdot|\mathbf{x}^q, \mathbf{v}^q, \mathbf{r}, \mathbf{z}_{<l}) || \pi_{\theta_l}(\cdot|\mathbf{v}^q, \mathbf{r}, \mathbf{z}_{<l})\right]$
  /* ELBO likelihood contribution update */
  ELBO $\leftarrow$ ELBO $+ \log \mathcal{N}\left(\mathbf{x}^q|\mu = \eta_\theta^g(\mathbf{u}_L), \sigma = \sigma_t\right)$

Algorithm S3: Generating a prediction from GQN.

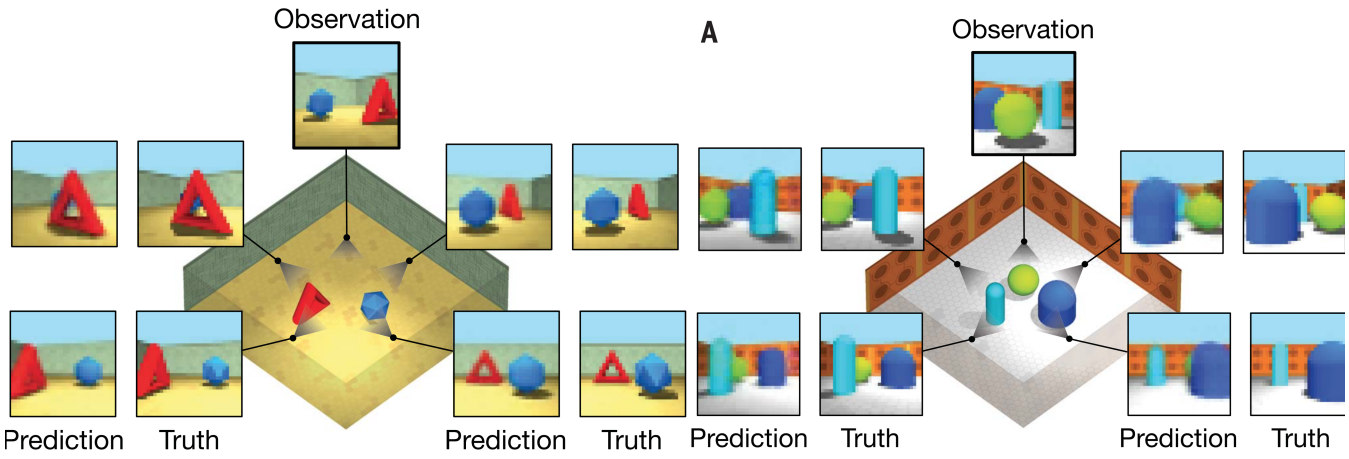
def *Generate*($\{(\mathbf{x}^k, \mathbf{v}^k)\}, \mathbf{v}^q$)**:**
  Output: Generated image sample $\hat{\mathbf{x}}^q$
  /* Scene encoder */
  $\mathbf{r} \leftarrow 0$
  for $k \leftarrow 0$ **in** $(M-1)$**:**
    $\hat{\mathbf{v}}^k \leftarrow (\mathbf{w}^k, \cos(\mathbf{y}^k), \sin(\mathbf{y}^k), \cos(\mathbf{p}^k), \sin(\mathbf{p}^k))$
    $\mathbf{r}^k \leftarrow \psi\left(\mathbf{x}^k, \hat{\mathbf{v}}^k\right)$
    $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{r}^k$
  /* Initial state */
  $(\mathbf{c}_0^g, \mathbf{h}_0^g, \mathbf{u}_0) \leftarrow (\mathbf{0}, \mathbf{0}, \mathbf{0})$
  for $l \leftarrow 0$ **in** $(L-1)$**:**
    /* Prior factor */
    $\pi_{\theta_l}(\cdot|\mathbf{v}^q, \mathbf{r}, \mathbf{z}_{<l}) \leftarrow \mathcal{N}\left(\cdot|\eta_\theta^\pi(\mathbf{h}_l^g)\right)$
    /* Prior sample */
    $\mathbf{z}_l \sim \pi_{\theta_l}(\cdot|\mathbf{v}^q, \mathbf{r}, \mathbf{z}_{<l})$
    /* State update */
    $\left(\mathbf{c}_{l+1}^g, \mathbf{h}_{l+1}^g, \mathbf{u}_{l+1}\right) \leftarrow C_\theta^g(\mathbf{v}^q, \mathbf{r}, \mathbf{c}_l^g, \mathbf{h}_l^g, \mathbf{u}_l, \mathbf{z}_l)$
  /* Image sample */
  $\hat{\mathbf{x}}^q \sim \mathcal{N}\left(\mathbf{x}^q|\mu = \eta_\theta^g(\mathbf{u}_L), \sigma = \sigma_t\right)$

Figure 5. Algorithm description of GQN.

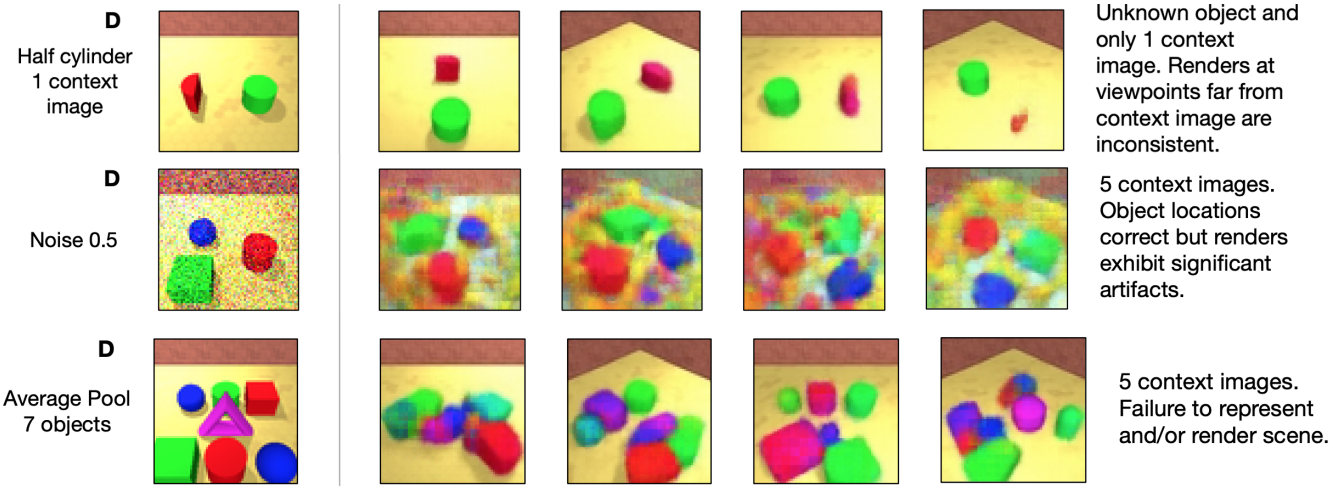
Note that on line 17 they missed a parameter $\mathbf{z}_l$ for the generator as suggested in eq. (S23).

4. IMPROVEMENTS AND FUTURE WORKS

4.1 Results and analysis

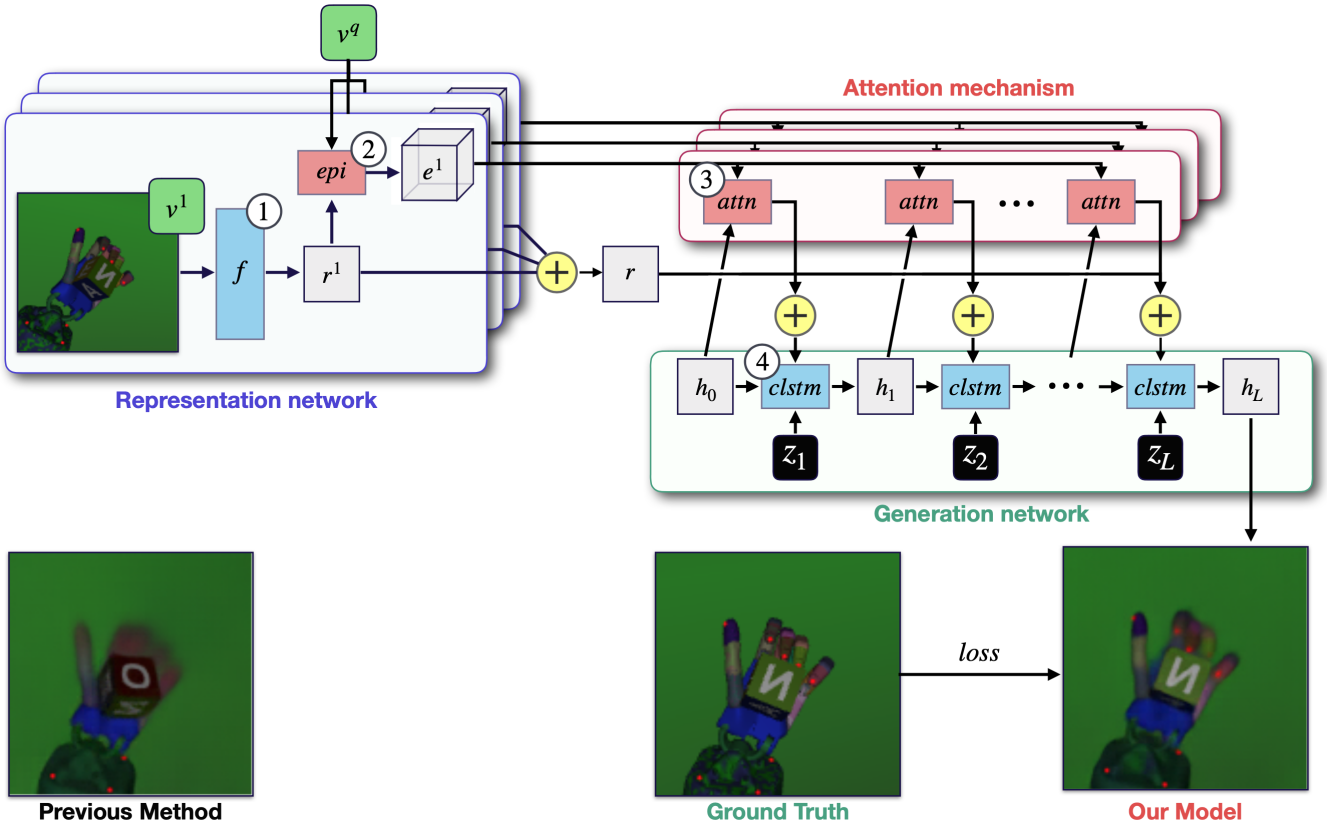


You can find more results and detailed experiment settings from this video and the Supplementary Material. Overall, we can see that with proper training, GQN is able to predict the unseen perspective of a fairly complex scene. However, some results shows that the paper's implementation of GQN would have poor performance with insufficient observations (context image), high noises or overcomplicated scene.



4.2 Improvements

For more complicated scene, [4] proposed an enhancement using epipolar geometry to model relationships between pixels across input observations even if they are spatially distant. The representation network **f** would now extract the scene representation **r** along with an epipolar representation **e**. In the generator network, **e** is used to calculate an *attention score* $\mathbf{a}_l$ that represents the weighted contribution to each spatial position of all the geometrically relevant features in the scene representation **r**.



Another potential problem with GQN is that the representation network does not scale well with the number of context images (input observations) provided. Because a simple aggregation function is used in 3.1. To improve the quality of the scene representation [2] proposed using reinforced learning techniques to select context images that will best describe the scene hence improving the quality of the representation vector **r**.

4.3 Applications and Outlook

The nature of GQN as another method that constructs 3D scene awareness from 2D images makes it naturally suitable for generating 3D models in cases where 3D capturing devices are not available. For example in street view construction, robot arm manipulation and automatic driving.

Another application of GQN could be in the art industry. 3D modeling can be hard work, it's quite difficult for 2D artists to work on 3D modeling. This paper [11] proposed that we can generate 3D models from 2D sketches using GQN, however the 2D drawings produced by artists do not exactly obey the physics law. One reason is that it's hard for human to be precise about physics when drawing with their hands. More importantly, artists often intentionally distort the physics law to emphasize specific features of the drawing or make the drawing aesthetically pleasing at different angles. [12]. These distortions can be subtle and natural to human perception that is either indistinguishable or is something that audiences can easily get used to. However, these discrepancies from the physics law will make accurately defining the scene difficult. With neural representations, we can skip the rigorous scene representation and traditional physics-based rendering techniques and generate 2D images directly from other drawings. This may even allow it to replicate specific art styles by extracting these laws that the artists keep in mind.

Finally, video-based prediction and interpolation is an important application of GQN. But sometimes the images generated by EQN can be a bit inconsistent making the resulting video looked 'jumpy'. This paper [3] resolved the issue by improving the consistency of EQN. With this in mind, we can think of another potential application in rendering with expensive rendering techniques, such as ray tracing. Some companies uses deep learning based upsampling techniques like DLSS from NVidia. With neural rendering, we can generate frames directly using the previous rendering results from the expensive rendering pipeline as context images and reducing the amount of frames we have to generate every second, which is similar to an interpolation that increases the rendering speed from a different direction.

Overall, I think EQN provided a brand new set of techniques that covers the entire rendering process from modeling to image generation and may have huge potentials to compensate for the disadvantages in traditional rendering techniques.

5. REFERENCES

[1] S. M. A. Eslami, D. Jimenez Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor, D. P. Reichert, L. Buesing, T. Weber, O. Vinyals, D. Rosenbaum, N. Rabinowitz, H. King, C. Hillier, M. Botvinick, D. Wierstra, K. Kavukcuoglu, and D. Hassabis, "Neural scene representation and rendering," SCIENCE, vol. 360, no. 6394, pp. 1204–1210, 2018.

[2] K. Chiang, "Using reinforcement learning to learn input viewpoints for scene representation," Master's thesis, EECS Department, University of California, Berkeley, May 2019.

[3] A. Kumar, S. M. A. Eslami, D. J. Rezende, M. Garnelo, F. Viola, E. Lockhart, and M. Shanahan, "Consistent generative query networks," *CoRR*, vol. abs/1807.02033, 2018.

[4] J. Tobin, W. Zaremba, and P. Abbeel, "Geometry-aware neural rendering," *NeurIPS*, vol. 32, 2019.

[5] "Computer graphics homework 4," 2021. https://billsun.dev/graphics/hw4.

[6] A. A. Soltani, H. Huang, J. Wu, T. D. Kulkarni, and J. B. Tenenbaum, "Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2511–2519, 2017.

[7] S. Lunz, Y. Li, A. Fitzgibbon, and N. Kushman, "Inverse graphics gan: Learning to generate 3d shapes from unstructured 2d data," 2020.

[8] V. Sitzmann, M. Zollhoefer, and G. Wetzstein, "Scene representation networks: Continuous 3d-structure-aware neural scene representations," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[9] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra, "Draw: A recurrent neural network for image generation," in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 1462–1471, PMLR, 07–09 Jul 2015.

[10] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 1278–1286, PMLR, 22–24 Jun 2014.

[11] M. N. Ramström, "Sketch to 3d model using generative query networks," 2019.

[12] E. Sugisaki, Y. Kazama, S. Morishima, N. Tanaka, and A. Sato, "Anime hair motion design from animation database," in *Proceedings of the 2006 International Conference on Game Research and Development*, CyberGames '06, (Murdoch, AUS), p. 33–40, Murdoch University, 2006.