Neural rendering

Neural radiance fields

Maciej Żelaszczyk

November 29, 2023

PhD Student in Computer Science
Division of Artificial Intelligence and Computational Methods
Faculty of Mathematics and Information Science

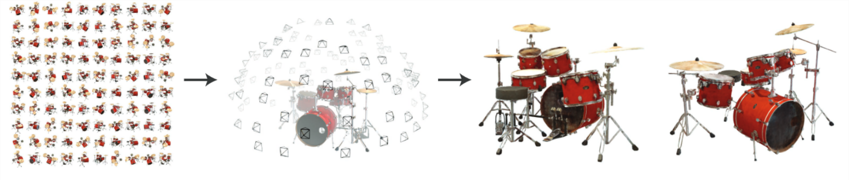
m.zelaszczyk@mini.pw.edu.pl

Warsaw University
of Technology

Problem statement

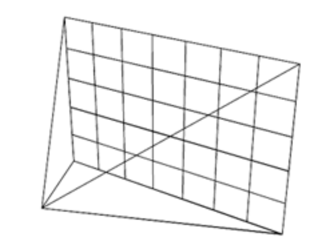
- We have a series of 2D images of a scene.
- We would like to be able to render this scene.
- This includes viewpoints not present in the original 2D images.

Problem statement



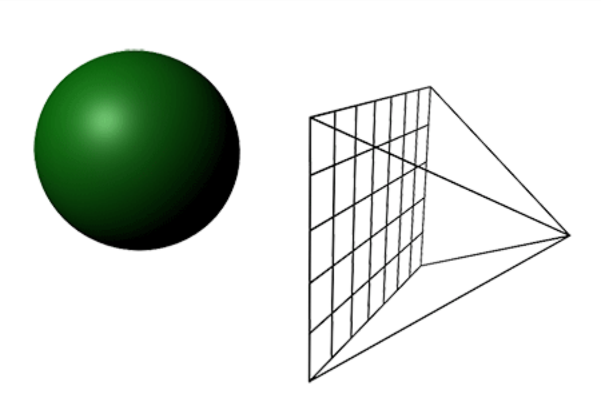
Source: [Mildenhall et al., 2020].

Ray tracing



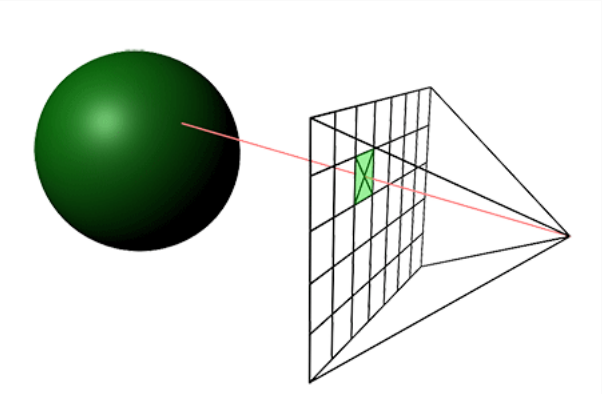
Source: An Overview of the Ray-Tracing Rendering Technique

Ray tracing



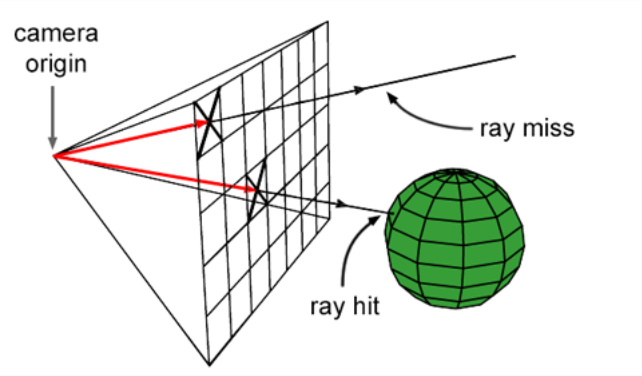
Source: Ray-Tracing: Generating Camera Rays

Ray tracing



Source: Ray-Tracing: Generating Camera Rays

Ray tracing



Source: An Overview of the Ray-Tracing Rendering Technique

Ray tracing

Setup with object detection:

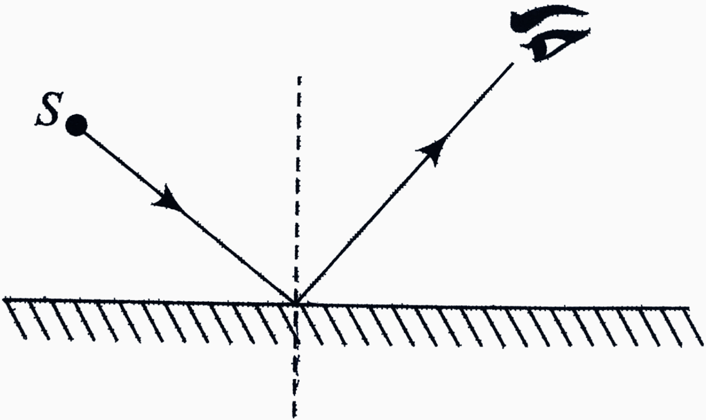
- Requires information on object positioning relative to one another.
- Allows to color specific pixels.
- Does not take into account light effects.

Ray tracing

Physical process of light reaching an observer:

- Light originates from a source.
- Multiple rays are cast from the source to the environment.
- These rays bounce of surfaces.
- Some of them reach an observer.
- The more light bounces off a specific surface point, the more illuminated it is to the observer.

Ray tracing



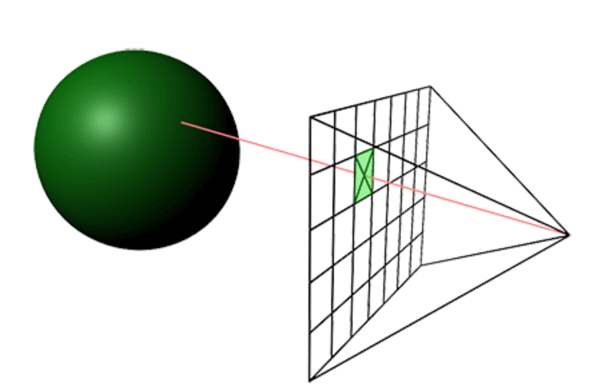
Source: Sarthaks

Ray tracing

Replicating the physical process:

- In principle should allow for realistic lighting effects.
- Leads to significant inefficiencies.
- A vast majority of rays cast from a light source will never reach the camera.

Ray tracing



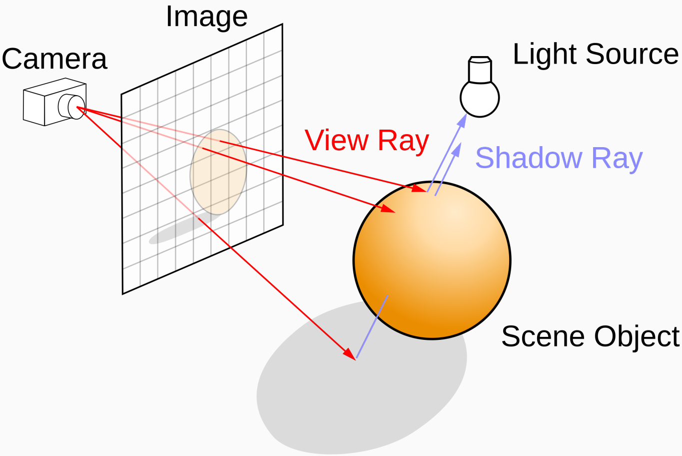
Source: Ray-Tracing: Generating Camera Rays

Ray tracing

Reverse the physical process:

- Cast rays from camera rather than from light source.
- Bounce those rays around in the environment.
- Proceed until rays reach a light source.
- The more direct the route between the camera and the light source, the more illuminated the point.
- In practice, this is more complicated: different light absorption of materials, ray splitting, etc.

Ray tracing



Source: Wikipedia

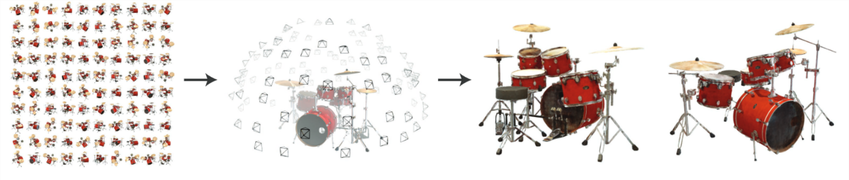
Image rendering

For our specific flavor of image rendering we require two components:

- World model.
- Ray casting procedure.

Let us adopt these abstract blocks to the stated problem.

Problem statement



Source: [Mildenhall et al., 2020].

Neural radiance fields

We will consider the following components:

- Volumetric scene function.
- Volumetric ray casting.

Volumetric scene function

To include lighting effects, each point in a 3D scene can be represented in terms of:

- Its *spatial location*.
- A *viewing direction* - the direction from which this point is observed.
- The spatial location can be represented as a 3D vector $\mathbf{x} = (x, y, z)$.
- The viewing direction can be represented by two angles (θ, ϕ).
- In practice, viewing direction is represented by a 3D unit vector $\mathbf{d}$.

Volumetric scene function

The volumetric scene function outputs the radiance at each point (x, y, z) emitted in direction (θ, ϕ), along with information how much radiance is present:

- Emitted color $\mathbf{c} = (r, g, b)$.
- Volume density σ.

Volumetric scene function

Continuous scene representation:

$$F_\Theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma) \tag{1}$$

Volumetric scene function

Practical considerations:

- Can be approximated by an MLP.
- Restrictions on internal structure of the MLP might be beneficial.
- In particular: density does not seem to depend on viewing direction, while the emitted color does.
- We may want to constrain the model to predict σ based on $\mathbf{x} = (x, y, z)$ alone.
- Predictions of the color $\mathbf{c} = (r, g, b)$ can still rely on $\mathbf{d}$.
- Operating directly on input turns out to underperform.
- Positional encodings of input perform significantly better.

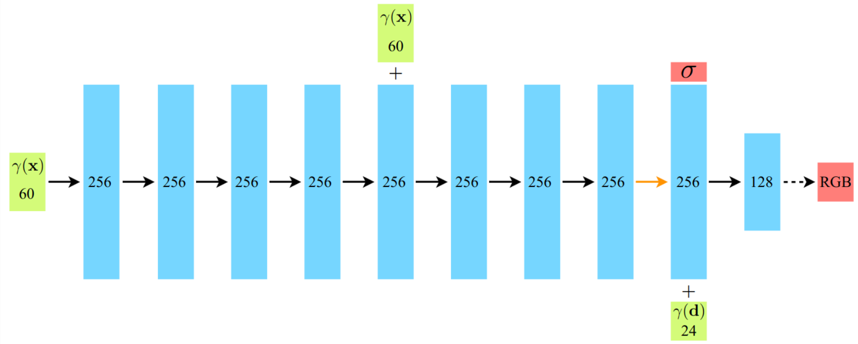
Volumetric scene function

Positional encoding:

$$\gamma(p) = \left(\sin\left(2^0 \pi p\right), \cos\left(2^0 \pi p\right), \ldots, \sin\left(2^{L-1} \pi p\right), \cos\left(2^{L-1} \pi p\right)\right) \tag{2}$$

- Applied to each element of $\mathbf{x}$ and $\mathbf{d}$.
- For $\mathbf{x}$, $L = 10$. Results in a 60-element encoding.
- For $\mathbf{d}$, $L = 4$. Results in a 24-element encoding.

Volumetric scene function



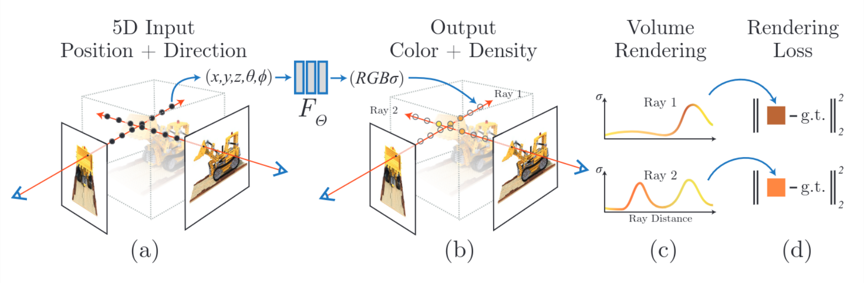
Source: [Mildenhall et al., 2020].

Volumetric ray casting

Given the volumetric scene function:

- It is possible to render the color of any ray passing through the scene.
- The volume density $\sigma(\mathbf{x})$ can be interpreted as the probability of a ray terminating at location $\mathbf{x}$.

Training procedure



Source: [Mildenhall et al., 2020].

Volumetric ray casting

The expected color of a camera ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ is:

Expected color of a camera ray:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt \tag{3}$$

where

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right) \tag{4}$$

and f_n and f_f are the near and far bounds, respectively. $T(t)$ can be interpreted as the probability that a ray travels from t_n to t.

Volumetric ray casting

Procedure for estimating the integral:

- Stratified sampling approach.
- Partition $[t_n; t_f]$ into N evenly-spaced bins.
- Draw one sample uniformly from each bin:

$$t_i \sim \mathcal{U}\left[t_n + \frac{i}{N}(t_f - t_n); t_n + \frac{i-1}{N}(t_f - t_n)\right] \tag{5}$$

Volumetric ray casting

Use quadrature rule to perform actual estimation:

Estimated color of a camera ray:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \left(1 - \exp\left(-\sigma_i \delta_i\right)\right) \mathbf{c}_i \tag{6}$$

where

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right) \tag{7}$$

and $\delta_i = t_{i+1} - t_i$ is the distance between adjacent samples.

Volumetric ray casting

The proposed sampling scheme has a major drawback:

- It does not take the positioning of objects into account.
- Free space and occluded objects are sampled repeatedly.
- They do not actually contribute to the image.
- To mitigate this, hierarchical sampling can be used.

Volumetric ray casting

Improved sampling scheme:

- Instead of one, apply two networks: *coarse* and *fine*.
- Sample set of N_c locations using the standard stratified sampling approach.
- Evaluate the *coarse* network at these locations.
- Rewrite colors from the *coarse* network:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N_c} w_i \mathbf{c}_i \tag{8}$$

$$w_i = T_i \left(1 - \exp\left(-\sigma_i \delta_i\right)\right) \tag{9}$$

Volumetric ray casting

Improved sampling scheme:

- Normalize: $\hat{w}_i = w_i / \sum_{j=1}^{N_c} w_j$.
- We now have a piecewise-constant PDF along the ray.
- Sample set of N_f locations from this distribution using inverse transform sampling.
- Evaluate the *fine* network the union of $N_c + N_f$ sample locations.
- Compute the final rendered color of the ray $\hat{C}_f(\mathbf{r})$ using the standard ray casting approach but for all $N_c + N_f$ samples.
- This guides the sampling procedure toward regions where we already expect visible content.

Training procedure

Optimize network:

- Use a dataset of RGB images of a specific scene.
- Use *structure-from-motion* to estimate camera poses, bounds, etc. for this scene.
- One possibility is to use COLMAP [Schonberger and Frahm, 2016].
- Sample a batch of pixels from all images in the dataset.
- Each pixel can be associated with a ray through the scene.
- Hierarchically sample locations along the rays.
- Use volumetric rendering to determine the color of each ray.

Training procedure

Optimize network:

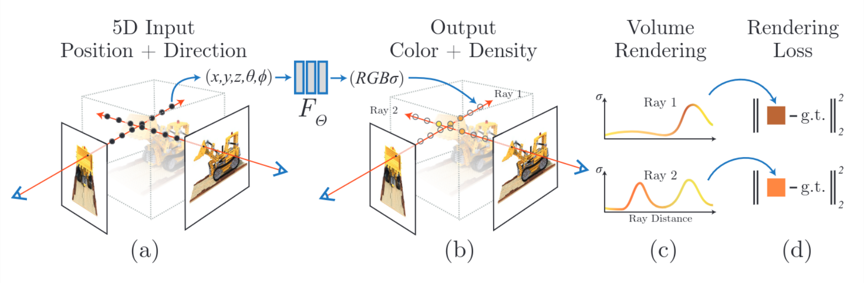
- Since we have a ground truth pixel, we can now calculate the loss:

Loss function:

$$\mathcal{L} = \sum_{\mathbf{r} \in \mathcal{R}} \left[\left\| \hat{C}_c(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 \right] \qquad (10)$$

where $\mathcal{R}$ is the batch of rays $C(\mathbf{r})$ is the ground truth, $\hat{C}_c(\mathbf{r})$ is the *coarse* color prediction and $\hat{C}_f(\mathbf{r})$ is the *fine* color prediction for a given ray $\mathbf{r}$.

Training procedure



Source: [Mildenhall et al., 2020].

Training procedure

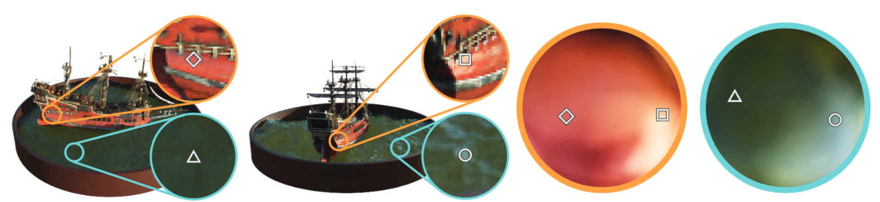
Practical considerations:

- Batch size $= 4096$.
- Numbers of sampled locations: $N_c = 64, N_f = 128$.
- Optimized with Adam. Inital learning rate $= 5 \times 10^{-4}$. Exponantial decay to 5×10^{-5}. Other hyperparameters left at defaults: $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-7}$.
- Steps to convergence: $100 - 300$k.
- Hardware: NVIDIA V100 GPU.
- Wall-clock time: $1 - 2$ days.

Training procedure

Datasets:

- Synthetic:
  - DeepVoxels [Sitzmann et al., 2019]: 4 Lambertian objects with simple geometry; 512×512 pixels; for each scene, viewpoints sampled on the upper hemisphere (479 for training and 1000 for testing).
  - Own dataset: 8 non-Lambertian objects with complicated geometry; 800×800 pixels; viewpoints sampled on the upper hemisphere for 6 scenes, on the full sphere for 2 scenes; for each scene, 100 viewpoints for training and 200 for testing.
- Real:
  - New dataset: 8 complex real-world scenes, recorded with handheld phones, 5 from the LLFF dataset [Mildenhall et al., 2019], 3 newly captured; 1008×756 pixels; $20 - 62$ images per scene, $1/8$ used for testing.



(a) View 1 (b) View 2 (c) Radiance Distributions

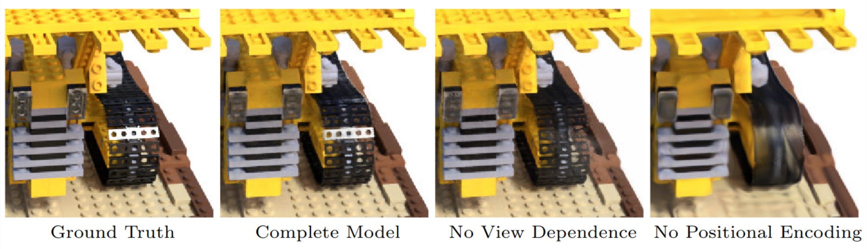
Source: [Mildenhall et al., 2020].

Results

Which components are crucial for achieving favorable results?

- Positional encoding.
- Hierarchical sampling.

Results



Source: [Mildenhall et al., 2020].

Considerations

- Why would you use a neural network to represent a scene?
  - Quality of generated scenes.
  - Size of representation. LLFF [Mildenhall et al., 2019] generates a 3D voxel grid of $>$ 15GB. For the same scene, NeRF requires 5MB for network weights.
- Is the training slow?
  - Relatively. NeRF takes at least 12 hours to train for one scene. LLFF takes around 10 minutes to generate a scene.
  - Ongoing work to cut this down.
- Is inference slow?
  - Initially, yes. Order of magnitude: 30 seconds per frame.
  - Massive improvement in subsequent works. Speeds $>$ 100 FPS are now achievable [Kerbl et al., 2023].
- It is still one scene per network?
  - Yes. Ongoing work on making this more general, e.g. using hyper networks [Lorraine et al., 2023, Bao et al., 2023].

Bao, Y., Ding, T., Huo, J., Li, W., Li, Y., and Gao, Y. (2023).
Insertnerf: Instilling generalizability into nerf with hypernet modules.

Kerbl, B., Kopanas, G., Leimkühler, T., and Drettakis, G. (2023).
3d gaussian splatting for real-time radiance field rendering.
ACM Transactions on Graphics, 42(4).

Lorraine, J., Xie, K., Zeng, X., Lin, C.-H., Takikawa, T., Sharp, N., Lin, T.-Y., Liu, M.-Y., Fidler, S., and Lucas, J. (2023).
Att3d: Amortized text-to-3d object synthesis.
In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 17946–17956.

Mildenhall, B., Srinivasan, P. P., Ortiz-Cayon, R., Kalantari, N. K., Ramamoorthi, R., Ng, R., and Kar, A. (2019).
Local light field fusion: Practical view synthesis with prescriptive sampling guidelines.
ACM Trans. Graph., 38(4).

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020).
Nerf: Representing scenes as neural radiance fields for view synthesis.
In *ECCV*.

Schonberger, J. L. and Frahm, J.-M. (2016).
Structure-from-motion revisited.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Sitzmann, V., Thies, J., Heide, F., Niessner, M., Wetzstein, G., and Zollhofer, M. (2019).
Deepvoxels: Learning persistent 3d feature embeddings.
In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.