AutoInt: Automatic Integration for Fast Neural Volume Rendering

David B. Lindell, Julien N. P. Martel, Gordon Wetzstein
Stanford University
{lindell, jnmartel, gordon.wetzstein}@stanford.edu

Abstract

Numerical integration is a foundational technique in scientific computing and is at the core of many computer vision applications. Among these applications, neural volume rendering has recently been proposed as a new paradigm for view synthesis, achieving photorealistic image quality. However, a fundamental obstacle to making these methods practical is the extreme computational and memory requirements caused by the required volume integrations along the rendered rays during training and inference. Millions of rays, each requiring hundreds of forward passes through a neural network are needed to approximate those integrations with Monte Carlo sampling. Here, we propose automatic integration, a new framework for learning efficient, closed-form solutions to integrals using coordinate-based neural networks. For training, we instantiate the computational graph corresponding to the derivative of the coordinate-based network. The graph is fitted to the signal to integrate. After optimization, we reassemble the graph to obtain a network that represents the antiderivative. By the fundamental theorem of calculus, this enables the calculation of any definite integral in two evaluations of the network. Applying this approach to neural rendering, we improve a tradeoff between rendering speed and image quality: improving render times by greater than 10× with a tradeoff of reduced image quality.

1. Introduction

Image-based rendering and novel view synthesis are fundamental problems in computer vision and graphics (e.g., [5, 54]). The ability to interpolate and extrapolate a sparse set of images depicting a 3D scene has broad applications in entertainment, virtual and augmented reality, and many other applications. Emerging neural rendering techniques have recently enabled photorealistic image quality for these tasks (see Sec. 2).

Although state-of-the-art neural volume rendering techniques offer unprecedented image quality, they are also extremely slow and memory inefficient [37]. This is a fundamental obstacle to making these methods practical. The primary computational bottleneck for neural volume rendering is the evaluation of integrals along the rendered rays during training and inference required by the volume rendering equation [33]. Approximate integration using Monte Carlo sampling is typically used for this purpose, requiring hundreds of forward passes through the neural network representing the volume for each of the millions of rays that need to be rendered for a single frame. Here, we develop a general and efficient framework for approximate integration. Applied to the specific problem of neural volume rendering, our framework improves a tradeoff between rendering speed and image quality, allowing a greater than 10× speedup in the rendering process, though with a reduction in image quality.
Our integration framework builds on previous work demonstrating that coordinate-based networks (sometimes also referred to as implicit neural representations) can represent signals (e.g., images, audio waveforms, or 3D shapes) and their derivatives. That is, taking the derivative of the coordinate-based network accurately models the derivative of the original signal. This property has recently been shown for coordinate-based networks with periodic activation functions [51], but we show that it also extends to a family of networks with different nonlinear activation functions (Sec 3.4 and supplemental).

We observe that taking the derivative of a coordinate-based network results in a new computational graph, a “grad network”, which shares the parameters of the original network. Now, consider that we use as our network a multilayer perceptron (MLP). Taking its derivative results in a grad network which can be trained on a signal that we wish to integrate. By reassembling the grad network parameters back into the original MLP, we construct a neural network that represents the antiderivative of the signal to integrate.

This procedure results in a closed-form solution for the antiderivative, which, by the fundamental theorem of calculus, enables the calculation of any definite integral in two evaluations of the MLP. Inspired by techniques for automatic differentiation (AutoDiff), we call this procedure automatic integration or AutoInt. Although the mechanisms of AutoInt and AutoDiff are very different, both approaches enable the calculation of integrals or derivatives in an automated manner that does not rely on traditional numerical techniques, such as sampling or finite differences.

The primary benefit of AutoInt is that it allows evaluating arbitrary definite integrals quickly by querying the network representing the antiderivative. This concept could have important applications across science and engineering; here, we focus on the specific application of neural volume rendering. For this application, efficiently evaluating integrals amounts to accelerating rendering (i.e., inference) times, which is crucial for making these techniques more competitive with traditional real-time graphics pipelines. However, our framework still requires a slow training process to optimize a network for a given set of posed 2D images.

Specifically, our contributions include the following.

- We introduce a framework for automatic integration that learns closed-form integral solutions. To this end, we explore new network architectures and training strategies.
- Using automatic integration, we propose a new model and parameterization for neural volume rendering that is efficient in computation and memory.
- We improve a tradeoff between neural rendering speed and image quality, demonstrating rendering rates that are an order of magnitude faster than previous implementations [37], though with a reduction in image quality.

2. Related Work

Neural Rendering. Over the last few years, end-to-end differentiable computer vision pipelines have emerged as a powerful paradigm wherein a differentiable or neural scene representation is optimized via differentiable rendering with posed 2D images (see e.g., [56] for a survey). Neural scene representations often use an explicit 3D proxy geometry, such as multi-plane [15, 36, 62] or multi-sphere [2, 4] images or a voxel grid of features [31, 52]. Explicit neural scene representations can be rendered quickly, but they are fundamentally limited by the large amount of memory they consume and thus may not scale well.

As an alternative, coordinate-based networks, or implicit neural representations, have been proposed as a continuous and memory-efficient approach. Here, the scene is parameterized using neural networks, and 3D awareness is often enforced through inductive biases. The ability to represent details in a scene is limited by the capacity of the network architecture rather than the resolution of a voxel grid, for example. Such representations have been explored for modeling shape parts [16, 17], objects [3, 6, 10, 19, 25, 27, 30, 34, 35, 40, 42, 43, 49, 53, 60], or scenes [14, 21, 29, 37, 45, 51]. Coordinate-based networks have also been explored in the context of generative frameworks [7, 9, 20, 38, 39, 50].

The method closest to our application is neural radiance fields (NeRF) [37]. NeRF is a neural rendering framework that combines a volume represented by a coordinate-based network with a neural volume renderer to achieve state-of-the-art image quality for view synthesis tasks. Specifically, NeRF uses ReLU-based multilayer perceptrons (MLPs) with a positional encoding strategy to represent 3D scenes. Rendering an image from such a representation is done by evaluating the volume rendering equation [33], which requires integrating along rays passing through the neural volume parameterized by the MLP. This integration is performed using Monte Carlo sampling, which requires hundreds of forward passes through the MLP for each ray. However, this procedure is extremely slow, requiring days to train a representation of a single scene from multi-view images. Rendering a frame from a pre-optimized representation requires tens of seconds to minutes.

Here, we leverage automatic integration, or AutoInt, to significantly speed up the evaluation of integrals along rays. AutoInt reduces the number of network queries required to evaluate integrals (e.g., using Monte Carlo sampling) from hundreds to just two, greatly speeding up inference for neural rendering.
AutoInt builds the grad network

AutoInt evaluates the integral

Instantiate the integral network

AutoInt optimizes the grad network

AutoInt evaluates the integral


AutoInt for Neural Integration

In this section, we introduce a fundamentally new approach to compute and evaluate antiderivatives and definite integrals of coordinate-based neural networks.

3.1. Principles

We consider a coordinate-based network, i.e., a neural network with parameters \( \theta \) mapping low-dimensional input coordinates to a low-dimensional output \( \Phi_{\theta} : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_{out}} \). We assume this network admits a (sub)-gradient with respect to its input \( x \in \mathbb{R}^{d_{in}} \), and we denote by \( \Psi_{\theta} = \partial \Phi_{\theta} / \partial x_i \) its derivative with respect to the coordinate \( x_i \). Then, by the fundamental theorem of calculus we have that

\[
\Phi_{\theta}(x) = \int \frac{\partial \Phi_{\theta}}{\partial x_i}(x) \, dx_i = \int \Psi_{\theta}(x) \, dx_i. \tag{1}
\]

This equation relates the coordinate-based network \( \Phi_{\theta} \) to its partial derivative \( \Psi_{\theta} \) and, hence, \( \Phi_{\theta} \) is an antiderivative of \( \Psi_{\theta} \).

A key idea is that the partial derivative \( \Psi_{\theta} \) is itself a coordinate-based network, mapping the same low-dimensional input coordinates \( x \in \mathbb{R}^{d_{in}} \) to the same low-dimensional output space \( \mathbb{R}^{d_{out}} \). In other words, \( \Psi_{\theta} \) is a different neural network that shares its parameters \( \theta \) with \( \Phi_{\theta} \) while also satisfying Equation 1. Now, rather than optimizing the coordinate-based network \( \Phi_{\theta} \), we optimize \( \Psi_{\theta} \) to represent a target signal, and we reassemble the optimized parameters (i.e., weights and biases) \( \theta \) to form \( \Phi_{\theta} \).

As a result, \( \Phi_{\theta} \) is a network that represents an antiderivative of \( \Psi_{\theta} \). We call this procedure of training \( \Psi_{\theta} \) and reassembling \( \theta \) to construct the antiderivative automatic integration. How to reassemble \( \theta \) depends on the network architecture used for \( \Phi_{\theta} \), and is addressed in the next section.

3.2. The Integral and Grad networks

Coordinate-based neural networks are usually formed from multilayer perceptron (MLP), or fully connected, architectures:

\[
\Phi_{\theta}(x) = W_n (\phi_{n-1} \circ \phi_{n-2} \circ \cdots \circ \phi_0)(x), \tag{2}
\]

with \( \phi_k : \mathbb{R}^{M_k} \rightarrow \mathbb{R}^{N_k} \) being the \( k \)-th layer of the neural network defined as \( \phi_k(y) = N_{L_k} (W_k y + b_k) \) using the parameters \( \theta = \{ W_k \in \mathbb{R}^{N_k \times M_k}, b_k \in \mathbb{R}^{M_k} \}, \forall k \} \) and the nonlinearity \( N_L \), which is a function applied point-wise to all the elements of a vector.

The computational graph of a 3-hidden-layer MLP representing \( \Phi_{\theta} \) is shown in Figure 2. Operations are indicated as nodes and dependencies as directed edges. Here, the arrows of the directed edges point towards nodes that must be computed first.

For this MLP, the form of the network \( \Psi_{\theta} = \partial \Phi_{\theta} / \partial x_i \) can be found using the chain rule

\[
\Psi_{\theta}(x) = \hat{\phi}_{n-1} \circ (\phi_{n-2} \circ \cdots \circ \phi_0)(x) \circ \cdots \circ \hat{\phi}_1 \circ \phi_0(x) \circ W_0 \epsilon_i, \tag{3}
\]

where \( \circ \) indicates the Hadamard product, \( \hat{\phi}_k(y) = \psi_{NL}^T (W_{k-1} y + b_{k-1}) \) and \( \epsilon_i \in \mathbb{R}^{d_{in}} \) is the unit vector that has 0’s everywhere but at the \( i \)-th component. The

Integration Techniques. In general, integration is much more challenging than differentiation. Whereas automatic differentiation primarily builds on the chain rule, there are many different strategies for integration, including variable substitution, integration by parts, partial fractions, etc. Heuristics can be used to choose one or a combination of these strategies for any specific problem. Closed-form solutions to finding general antiderivatives exist only for a relatively small class of functions and, when possible, involve a rather complex algorithm, such as the Risch or Risch-Norman algorithm \([41, 47, 48]\). Perhaps the most common approach to computing integrals in practice is numerical integration, for example using Riemann sums, quadratures, or Monte-Carlo methods \([11]\). In these sampling-based methods, the number of samples trades off accuracy for runtime.

Since neural networks are universal function approximators, and are themselves functions, they can also be integrated analytically. Previous work has explored theory and connections between shallow neural networks and integral formulations for function approximation \([12, 22]\). Other work has derived closed-form solutions for integrals of simple single-layer or two-layer neural networks \([55, 58]\). As we shall demonstrate, our work is not limited to a fixed number of layers or a specific architecture. Instead, we directly train a grad network architecture for which the integral network is known by construction.
corresponding computational graph is shown in Figure 2. As we noted, despite having a different architecture (and vastly different number of nodes) the two networks share the same parameters. We refer to the network associated with $\Phi_\theta$ as the integral network and the neural network associated with $\Psi_i$ as the grad network. Homologous nodes in their graphs are shown in the same color. This color scheme explicitly shows how the grad network parameters are reassembled to create the integral network after training.

### 3.3. Evaluating Antiderivatives & Definite Integrals

To compute the antiderivative and definite integrals of a function $f$ in the AutoInt framework, one first chooses the specifics of the MLP architecture (number of layers, number of features, type of nonlinearities) for an integral network $\Phi_\theta$. The grad network $\Psi_i$ is then instantiated from this integral network based on AutoDiff. In practice, we developed a custom AutoDiff framework that traces the integral network and explicitly instantiates the corresponding grad network while maintaining the shared parameters (additional details in the supplemental). Once instantiated, parameters of the grad network are optimized to fit a signal of interest using conventional AutoDiff and optimization tools [44]. Specifically we optimize a loss of the form

$$\theta^* = \arg \min_\theta \mathcal{L} \left( \Psi_i^0(x), f(x) \right).$$

Here, $\mathcal{L}$ is a cost function that aims at penalizing discrepancies between the target signal $f(x)$ we wish to integrate and the coordinate-based network $\Psi_i^0$.

Once trained, the grad network approximates the signal, that is $\Psi^0_i \approx f(x)$, $\forall x$. Therefore, the antiderivative of $f$ can be calculated as

$$\int f(x) \, dx_i \approx \int \Psi_i^0(x) \, dx_i = \Phi_\theta^*(x).$$

This corresponds to evaluating the integral network at $x$ using weights $\theta^*$. Furthermore, any definite integral of the signal $f$ can be calculated using only two evaluations of $\Phi_\theta$, according to the Newton–Leibniz formula

$$\int_a^b f(x) \, dx_i = \Phi_\theta(b) - \Phi_\theta(a).$$

We also note that AutoInt extends to integrating high-dimensional signals using a generalized fundamental theorem of calculus, which we describe in the supplemental.

### 3.4. Example in Computed Tomography

In tomography, integrals are at the core of the imaging model: measurements are line integrals of the absorption of a medium along rays that go through it. In particular, in a parallel beam setup, assuming a 2D medium of absorption $f(x, y) \in \mathbb{R}_+$, measurements can be modeled as

$$s(\rho, \alpha) = \int_{t_n}^{t_f} f(x(t), y(t)) \, dt,$$

and $(x, y)$ is on the ray $(\rho, \alpha) \in [-1, 1] \times [0, \pi]$ by satisfying $x(t) \cos(\alpha) + y(t) \sin(\alpha) = \rho$ with $\alpha$ being the orientation of the ray and $\rho$ its eccentricity with respect to the origin as shown in Figure 3. The measurement $s$ is called a sinogram, and this particular integral is referred to as the Radon transform of $f$ [24].

The inverse problem of computed tomography involves recovering the absorption $f$ given a sinogram. Here, for illustrative purposes, we will look at a tomography problem in which a grad network is trained on a sparse set of measurements and the integral network is evaluated to produce unseen ones. Sparse-view tomography is a standard reconstruction problem [18], and this setup is analogous to the novel view synthesis problem we solve in Section 4.

We consider a dataset of measurements $\mathcal{D} = \{(\rho_i, \alpha_i, s(\rho_i, \alpha_i))\}_{i < D}$ corresponding to $D$ sparsely sampled rays. We train a grad network using the AutoInt framework. For this purpose, we instantiate a grad network $\Psi_\theta$ whose input is a tuple $(\rho, \alpha, t)$. It is trained to match the dataset of measurements

$$\theta^* = \arg \min_\theta \sum_{i < D} \left\| \left( \frac{1}{T} \sum_{j < T} \Psi_i^0(\rho_i, \alpha_i, t_j) \right) - s(\rho_i, \alpha_i) \right\|_2^2.$$ 

Thus, at training time, the grad network is evaluated $T$ times in a Monte Carlo fashion with $t_j \sim \mathcal{U}([t_n, t_f])$. At inference, just two evaluations of $\Phi_\theta$, yield the integral

$$s(\rho, \alpha) = \Phi_\theta^*(\rho, \alpha, t_f) - \Phi_\theta^*(\rho, \alpha, t_n).$$
Results in Figure 3 show that the two evaluations of the integral network Φθ can faithfully reproduce supervised measurements and generalize to unseen data. Generalization, however, depends on the type of nonlinearity used. We show that Swish [46] with normalized positional encoding (details in Sec. 5) generalizes well, and SIREN [51] fits the measurements better but fails to generalize to unseen views.

Note that both the nonlinearity NL and its derivative NL’ appear in the grad network architectures (Eq. (3) and Figure 2). This implies that integral networks with ReLU nonlinearities have step functions appearing in the grad network, possibly making training Ψθ difficult because of nodes with (constant) zero-valued derivatives. We explore several other nonlinearities here (with additional details in the supplemental), and show that Swish heuristically performs best in the grad networks used in our application. Yet, we believe the study of nonlinearities in grad networks to be an important avenue for future work.

4. Neural Volume Rendering

Combining volume rendering techniques with coordinate-based networks has proved to be a powerful technique for neural rendering and view synthesis [37]. Here, we briefly overview volume rendering and describe an approximate volume rendering model that enables our efficient rendering approach using AutoInt.

4.1. Volume Rendering

Classical volume rendering techniques are derived from the radiative transfer equation [8] with an assumption of minimal scattering in an absorptive and emissive medium [15, 33]. We adopt a rendering model based on tracing rays through the volume [23, 37], where the emission and absorption along camera rays produce color values that are assigned to rendered pixels.

The volume itself is represented as a high-dimensional function parameterized by position, $x \in \mathbb{R}^3$, and viewing direction $d$. We also define the camera rays that traverse the volume from an origin point $o$ to a ray position $r(t) = o + t \cdot d$. At each position in the volume, an absorption coefficient, $\sigma \in \mathbb{R}_+$, gives the probability per differential unit length that a ray is absorbed (i.e., terminates) upon interaction with an infinitesimal particle. Finally, an emissive radiance field $c = (r, g, b) \in [0, 1]^3$, describes the color of emitted light at each point in space in all directions.

Rendering from the volume requires integrating the emissive radiance along the ray while also accounting for absorption. The transmittance $T$ describes the net reduction from absorption from the ray origin to the ray position $r(t)$, and is given as

$$T(t) = \exp \left( - \int_{t_n}^{t} \sigma(r(s)) \, ds \right), \quad (10)$$

where $t_n$ indicates a near bound along the ray. With this expression, we can define the volume rendering equation (VRE), which describes the color $C$ of a rendered camera ray.

$$C(r) = \int_{t_n}^{t_f} T(t) \, \sigma(r(t)) \, c(r(t), d) \, dt. \quad (11)$$

Conventionally, the VRE is computed numerically by Riemann sums, quadratures, or Monte-Carlo methods [11], whose accuracy thus largely depends on the number of samples taken along the ray.

4.2. Approximate Volume Rendering for Automatic Integration

Automatic integration allows us to efficiently evaluate definite integrals using a closed-form solution for the antiderivative. However, the VRE cannot be directly evaluated with AutoInt because it consists of multiple nested integrations: the integration of radiance along the ray weighted by integrals of cumulative transmittance. We therefore choose to approximate this integral in piecewise sections that can each be efficiently evaluated using AutoInt. For $N$ piecewise sections along a ray, we give the approximate VRE and transmittance as

$$\tilde{C}(r) = \sum_{i=1}^{N} \tilde{\sigma}_i \tilde{e}_i \tilde{T}_i \delta_i, \quad \tilde{T}_i = \exp \left( - \sum_{j=1}^{i-1} \tilde{\sigma}_j \tilde{\delta}_j \right), \quad (12)$$

where

$$\tilde{\sigma}_i = \delta_i^{-1} \int_{t_{i-1}}^{t_i} \sigma(t) \, dt \quad \text{and} \quad \tilde{e}_i = \delta_i^{-1} \int_{t_{i-1}}^{t_i} c(t) \, dt,$$

and $\delta_i = t_i - t_{i-1}$ is the length of each piecewise interval along the ray. Equation 12 can also be viewed as a repeated alpha compositing operation with alpha values of $\tilde{\sigma}_i, \tilde{\delta}_i$. After some simplification and substitution into Equation 12 (see supplemental), we have the following expression for the piecewise VRE:

$$\tilde{C}(r) = \sum_{i=1}^{N} \delta_i^{-1} \int_{t_{i-1}}^{t_i} \sigma(t) \, dt \cdot \int_{t_{i-1}}^{t_i} c(t) \, dt$$

$$\cdot \prod_{j=1}^{i-1} \exp \left( - \int_{t_{j-1}}^{t_j} \sigma(s) \, ds \right). \quad (13)$$

While this piecewise expression is only an approximation to the full VRE, it enables us to use AutoInt to efficiently evaluate each piecewise integral over absorption and radiance. In practice, there is a tradeoff between improved computational efficiency and degraded accuracy of the approximation as the value of $N$ decreases. We evaluate this tradeoff in the context of volume rendering and learned novel view synthesis in Sec. 6.
5. Optimization Framework

We evaluate the piecewise VRE introduced in the previous section using an optimization framework overviewed in Figure 4. At the core of the framework are two MLPs that are used to compute integrals over values of $\sigma$ and $c$ as we detail in the following.

Network Parameterization. Rendering an image from the high-dimensional volume represented by the MLPs requires evaluating integrals along each ray $r(t)$ in the direction of $t$. Thus, the grad network should represent $\partial \Phi_\theta / \partial t$, the partial derivative of the integral network with respect to the ray parameter. In practice, the networks take as input the values that define each ray: $o$, $t$, and $d$. Then, positions along the ray are calculated as $x = o + t \cdot d$ and passed to the initial layers of the networks together with $d$. With this dependency on $t$, we use our custom AutoDiff implementation to trace computation through the integral network, define the computational graph that computes the partial derivative with respect to $t$, and instantiate the grad network.

Grad Network Positional Encoding. As demonstrated by Mildenhall et al. [37], a positional encoding on the input coordinates to the network can significantly improve the ability of a network to render fine details. We adopt a similar scheme, where each input coordinate is mapped into a higher dimensional space as using a function $\gamma(p) : \mathbb{R} \mapsto \mathbb{R}^{2L}$ defined as

$$\gamma(p) = (\sin(\omega_0 p), \cos(\omega_0 p), \cdots, \sin(\omega_{L-1} p), \cos(\omega_{L-1} p)), \quad (14)$$

where $\omega_i = 2^i \pi$ and $L$ controls the number of frequencies used to encode each input. We find that using this scheme directly in the grad network produces poor results because it introduces an exponentially increasing amplitude scaling into the coordinate encoding. This can easily be seen by calculating the derivative $\partial \gamma / \partial p = (\cdots, \omega_i \cos(\omega_i p), -\omega_i \sin(\omega_i p), \cdots)$. Instead, we use a normalized version of the positional encoding for the integral network, which improves performance when training the grad network:

$$\bar{\gamma}(p) = (\cdots, \omega_i^{-1} \sin(\omega_i p), \omega_i^{-1} \cos(\omega_i p), \cdots). \quad (15)$$

Predictive Sampling. While AutoInt is used at inference time, at training time, the grad network is optimized by evaluating the piecewise integrals of Equation 13 using a quadrature rule discussed by Max [33]:

$$\tilde{C}(r) = \sum_{i=1}^{N} \tilde{T}_i \left(1 - \exp(-\sigma_i \delta_i)\right) \tilde{e}_i. \quad (16)$$

We use Monte Carlo sampling to evaluate the integrals $\tilde{\sigma}_i$ and $\tilde{c}_i$ by querying the networks at many positions within each interval $\delta_i$.

However, some intervals $\delta_i$ along the ray contribute more to a rendered pixel than others. Thus, assuming we use the same number of samples per interval, we can improve sample efficiency by strategically adjusting the length of these intervals to place more samples in positions with large variations in $\sigma$ and $c$.

To this end, we introduce a small sampling network (illustrated in Figure 4), which is implemented as an MLP $\tilde{S}(o, d)$ that predicts interval lengths $\delta \in \mathbb{R}^N$. Then, we can calculate stratified samples along the ray by subdividing each interval $\delta_i$ into $M$ bins and calculating samples $t_{i,j} = 1, \ldots, M$ as $t_{i,j} \sim \mathcal{U}(t_{i-1} + \frac{j-1}{M} \delta_i, t_{i-1} + \frac{j}{M} \delta_i)$.
Fast Grad Network Evaluation. AutoInt can be implemented directly in popular optimization frameworks (e.g., PyTorch [44], Tensorflow [1]); however, training the grad network is generally computationally slow and memory inefficient. These inefficiencies stem from the two step procedure required to compute the grad network output at each training iteration: (1) a forward pass through the integral network is computed and then (2) AutoDiff calculates the derivative of the output with respect to the input variable of integration. Instead, we implemented a custom AutoDiff framework on top of PyTorch that parses a given integral network and explicitly instantiates the grad network modules with weight sharing (see Figure 2). Then, we evaluate and train the grad network directly, without the overhead of the additional per-iteration forward pass and derivative computation. Compared to the two-step procedure outlined above, our custom framework improves per-iteration training speed by a factor of 1.8 and reduces memory consumption by 15% for the volume rendering application. More details about our AutoInt implementation can be found in the supplemental, and our code is publicly available1.

Implementation Details. In our framework, a volume representation is optimized separately for each rendered scene. To optimize the grad networks, we require a collection of RGB images taken of the scene from varying camera positions, and we assume that the camera poses and intrinsic parameters are known. At training time, we randomly sample images from the training dataset, and from each image we randomly sample a number of rays. Then, we optimize the network to minimize the loss function

\[
\mathcal{L} = \sum_{\mathbf{r} \in \mathcal{R}} \| \mathbf{\tilde{C}}(\mathbf{r}) - \mathbf{C}(\mathbf{r}) \|^2, \tag{17}
\]

where \( \mathbf{C} \) is the ground truth pixel value for the selected ray.

In our implementation, we train the networks using PyTorch and the Adam optimizer [26] with a learning rate of \( 5 \times 10^{-4} \). The networks representing volume density and color each have 8 hidden layers with 256 hidden units, we use a batch size of 4 with 1024 rays sampled from each image, and we decay the learning rate by a factor of 0.2 every \( 10^5 \) iterations. Training and inference are performed using NVIDIA V100 GPUs. For the sampling network, we evaluate using \( M = 128/N \) samples within each piecewise interval for \( N \in \{2, 4, 8, 16, 32, 64\} \) (see Figure 5, supplemental) and find that using 8, 16, or 32 piecewise intervals produces acceptable results while achieving a significant computational acceleration with AutoInt. Finally, for the positional encoding, we use \( L = 10 \) and \( L = 4 \) for \( x \) and \( d \), respectively.

6. Results

We evaluate AutoInt for volume rendering on a synthetic dataset of scenes with challenging geometries and reflectance properties. We demonstrate that the approach allows an improved tradeoff between image quality and rendering speed for neural volume rendering. Rendering times are improved by greater than \( 10 \times \) compared to the state-of-the-art [37], though at reduced image quality.

Our training dataset consists of eight objects, each rendered from 100 different camera positions using the Blender Cycles engine [37]. For the test set, we evaluate on an additional 200 images. We compare AutoInt to two other baselines: Neural Radiance Fields (NeRF) [37] and Neural

---

1https://github.com/computational-imaging/automatic-integration
Volumes [31]. NeRF uses a similar architecture and Monte Carlo sampling with the full volume rendering model, rather than our piecewise approximation and AutoInt. Neural Volumes is a voxel-based method that encodes a deep voxel grid representation of a scene using a convolutional neural network. Novel views are rendered by applying a learned warping operator to the voxel grid and sampling voxel values by marching rays from the camera position.

In Table 1 we report the peak signal-to-noise ratio (PSNR) averaged across all scenes and test images. AutoInt outperforms Neural Volumes quantitatively, while achieving a greater than 10× improvement in render time relative to NeRF, though with a tradeoff in image quality. Increasing the number of piecewise sections in the approximate VRE improves render quality at the cost of computation.

We evaluate the effect of the sampling network and the number of sections in the approximate VRE in Figure 5 for the Lego scene. Using the sampling network improves performance and sample efficiency by allocating more sections in regions with large variations in the volume density.

In Table 2 we show the effect of the VRE approximation and grad network architecture on render quality of the Lego scene. Using the full VRE achieves similar performance to the approximate, piecewise VRE with 32 sections. We attribute most of the difference in performance between our method and NeRF to the regularized, tree-like structure of the grad network, which is constrained by weight sharing between the branches (see Figure 2). While evaluating NeRF with fewer samples along each ray reduces computation, rendering quality degrades significantly compared to using AutoInt with the same number of samples.

We also show qualitative results in Figure 6 for the Materials and Drums scenes. Again, the quality of the rendered images improves as the number of sections increases. In the Materials scene (Figure 6), the proposed technique exhibits fewer artifacts compared to Neural Volumes. AutoInt also shows improved modeling of view-dependent effects in the Drums scene relative to Neural Volumes and NeRF (e.g., specular highlights on the symbols). We show additional results on captured scenes from the Local Light Field Fusion and DeepVoxels datasets [36, 52] in the supplemental.

7. Discussion

In this work, we introduce a new framework for numerical integration in the context of coordinate-based neural networks. Applied to neural volume rendering, AutoInt enables improvements to computational efficiency by learning closed-form solutions to integrals. Although these computational speedups currently come with a tradeoff to image quality, the method takes steps towards efficient learned integration using deep network architectures. Our approach is analogous to conventional methods for fast evaluation of the VRE; for example, methods based on shear-warping [28] and the Fourier projection-slice theorem [32, 57]. Similar to our method, these techniques use approximations (e.g., with sampling and interpolation) that trade off image quality with computationally efficient rendering. Additionally, we believe our approach is compatible with recent work that aims to speed up volume rendering by pruning areas of the volume that do not contain the rendered object [29].

A key idea of AutoInt is that an integral network can be automatically created after training a corresponding grad network. Thus, exploring new grad network architectures that enable fast training with rapid convergence is an important and promising direction for future work. Moreover, we believe that AutoInt will be of interest to a wide array of application areas beyond computer vision, especially for problems related to inverse rendering, sparse-view tomography, and compressive sensing.
References


