PAVE: An In Situ Framework for Scientific Visualization and Machine Learning Coupling

1st Samuel Leventhal
University of Utah School of Computing
Scientific Computing and Imaging Institute
Salt Lake City, UT., USA
samlev@cs.utah.edu

2nd Mark Kim
Oak Ridge National Laboratory
Oak Ridge TN., USA
kimmb@ornl.gov

3rd David Pugmire
Oak Ridge National Laboratory
Oak Ridge TN., USA
pugmire@ornl.gov

***Abstract*—Machine learning (ML) has emerged as a tool for understanding data at scale. However, this new methodology comes at a cost because even more HPC resources are required to generate ML algorithms. In addition to the compute resources required to develop ML algorithms, ML does not sidestep one of the biggest challenges on leading-edge HPC systems: the increasing gap between compute performance and I/O bandwidth. This has led to a strong push towards *in situ* designs by processing data as it is generated and developing strategies to mitigate the I/O bottleneck. Unfortunately, there are no *in situ* frameworks dedicated to coupling scientific visualization and ML at scale to develop ML algorithms for scientific visualization.**

To address the ML and *in situ* visualization gap, we introduce PAVE. PAVE is an *in situ* framework which addresses the data management needs between visualisation and machine learning tasks. We demonstrate our framework with a case study that accelerates physically-based light rendering, path-tracing, through the use of a conditional Generative Adversarial neural Network (cGAN). PAVE couples the training over path-traced images resulting in a generative model able to produce scene renderings with accurate light transport and global illumination of a quality comparable to offline approaches in a more efficient manner.

***Index Terms*—VTKm, neural networks, generative adversarial network, in situ, PyTorch, path-tracing**

I. Introduction

Machine learning (ML) has become a significant driving force within the HPC community. New ML algorithms are being developed to analyze the data deluge coming from simulation codes, "Big Data" problems, as well as scientific instruments. Unfortunately, I/O bandwidth continues to lag behind compute [1]. However, *in situ* processing [2], which processes the data where it is generated, has stepped in to cope with this I/O bottleneck. These *in situ* frameworks have become more common in the modern HPC environment.

Machine learning has a cost, though. Traditionally, a researcher develops an algorithm in combination with data to produce a scientific visualization output such as a rendered image or a triangular mesh (Fig. 1). Developing ML algorithms is the inverse of the traditional programming model: data is processed with traditional output (simulation data, high-quality rendered images, etc) to generate ML algorithms. The construction of a ML algorithm is then the byproduct of coupling data with the output of a learning approach to converge on a learned algorithmic model while traditional

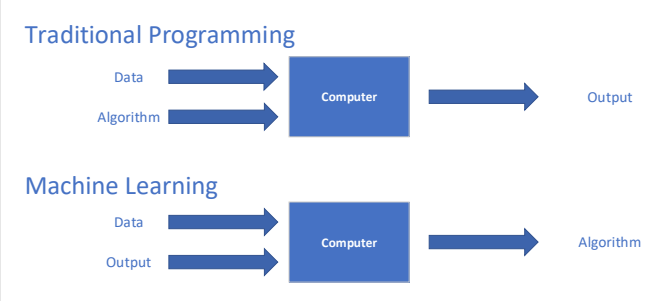


Fig. 1: Traditional programming of algorithm and data coupled to produce output vs machine learning methodology of data being coupled with it's output to produce a trained algorithmic model.

programming's focus is on generating output from pre-existing algorithms using data. Significant amount of computation as well as copious amounts of traditional output can be required to construct sophisticated ML algorithms. To perform this coupling efficiently at scale *in situ* processing is required but currently there are no frameworks specifically tailored to *in situ* scientific visualization and ML.

To bridge this divide we present PAVE, an *in situ* framework for coupling scientific visualization and machine learning. The purpose of this framework is to offer a solution which provides the means necessary to couple machine learning implementations at scale and *in situ* to aid or be the basis of scientific visualisation tasks. We accomplish this by providing read, write and in place data transfer functionality between learning models and visualisation task implementations. To our knowledge, this is the first scientific visualization-machine learning *in situ* framework that easily allows a user to implement visualization based machine learning tasks at scale.

Further, we present a case study of coupling a path-tracer, a physically-based light rendering model, with a neural network to build a filter for a more efficient, but accurate light transport. The case study utilizes VTK-m, a toolkit for massively threaded architectures for scientific visualization and Python, an increasingly popular language within the machine learning community due to robust libraries available for neural networks such as PyTorch. The resulting work accomplishes

this combination by utilising VTK-m to construct a path trace rendering tool able to fluidly and efficiently communicate to a cGAN by means of PAVE during training. The resulting generative model serves as a real-time filter for rendering globally illuminated images which accurately approximate diffuse indirect illumination and soft shadows with quality comparable to offline approaches.

Though we demonstrate the use of PAVE with a path tracer and cGAN, PAVE offers support for numerous potential applications which aim to design a scalable coupling of visualization and machine learning such as, for example, the use of generative networks to create synthetic data used to expand training sets or emulate scientific data or for the application of learning approaches in situ to scientific visualizations being rendered.

II. Related Work

A. In situ Data Management

As output sizes of simulations have grown to petabytes of data, new strategies are required to handle the volumes of data generated. *In situ* analysis and visualization frameworks have emerged to handle this deluge of data, dependent on the needs and requirements of the HPC applications [3]–[6]. Some frameworks provide a generic data interface [7], [8] with triggers [9]. Others focus on *in situ*, IO-oriented approaches [10], which enables new paradigms for *in situ* performance [11]. Most of these frameworks for *in situ* processing are tailored towards coupling simulation data and visualization and analysis, but none are particularly focused on coupling scientific visualization and machine learning.

B. Heterogeneous Visualization

Heterogeneous environments in HPC require new solutions for extracting the most performance. The accelerator, such as GPGPU, in these heterogeneous environments have thousands of concurrent threads with various APIs, such as CUDA [12] or Thread Building Blocks [13]. Thrust [14] is a proprietary API for parallel primitives programming [15] on Nvidia hardware. VTK-m is a data parallel primitive toolkit for scientific visualization [16] which has a general parallel programming model with multiple backends to facilitate porting to different heterogeneous systems.

C. Path Tracing

Real time, true to life quality renderings of light transport through path tracing [17] remains an active area of research with a number of various approaches. Off-line path tracing has become the standard rendering approach in the movie industry because of the high quality of the image generated [18], but is unfortunately slow. To preserve real-time rates, previous works have stored precomputed radiance transfers for light transport as spherical functions within a fixed scene geometry which are then adjusted for varied light and camera perspective through projections within a basis of spherical harmonics [19]. Similarly, Light Propagation Volumes have been used to iteratively propagate light between consecutive grid positions to emulate single-bounce indirect illumination [20]. More recently, deep neural networks have been employed as a learned look up table for real-time rates with offline quality. With the use of convolutional neural networks Deep Shading is able to translate screen space buffers to into desired screen space effects such as indirect light, depth or motion blur. Similar to the methodology implemented in this work, Deep Illumination uses a conditional adversarial network (cGAN) to train a generative network with screen space buffers allowing for a trained network able to produce accurate global illumination with real-time rates at offline quality through a "one network for one scene" setting [21]. A comprehensive overview of physically-based rendering is beyond the scope of this work, for a more thorough overview of physically-based rendering we suggest [22] and [23] for sampling and reconstruction of Monte Carlo rendering.

III. Coupling Scientific Visualization and Machine Learning

PAVE is an *in situ* framework for coupling scientific visualization and machine learning. The resulting framework consists of two core components, visualization output coupled to a machine learning application. A scientific visualization application is coupled through the C++ interface, and the output is sent through PAVE to the machine learning application to construct a new algorithm (Fig. 1). PAVE then allows each task to communicate results among each other seamlessly. The scientific visualisation task developed by the user can then provide resulting visualisations to a separately developed learning model as input or employ a learning model within the visualisation task.

A. User Provided Visualisation

PAVE couples visualization and machine learning through an *in situ* framework that supports *in situ*, in transit, and post processing, depending on the configuration. It provides a unified interface which allows, among other things, memory-to-memory transport between applications during the I/O phase of a simulation, i.e. in transit. With the unified interface, applications can be coupled either memory-to-memory or through the filesystem. The results for the scalable, *in situ* scientific visualization task designed by the user, are passed to a learning model through a single interface and the visualization can remain fully scalable to distributed systems because of PAVE. For this same reason in the provided example in section IV-D we chose VTK-m arrays as the data structures of the visualisation task.

For the C++ scientific visualization interface, PAVE's design pattern is `initialize`, `save`, and optionally `flush`. The PAVE class is `initialized` with a name for the training set that will be generated by the visualization tool, that will be used with the machine learning application. As each datum (image, text, etc.) is generated, it is `saved` to the training set. PAVE buffers the data which allows the `save` task to be asynchronous. This enables overlapping I/O operations with visualization tasks. Optionally, there is a `flush` operation,

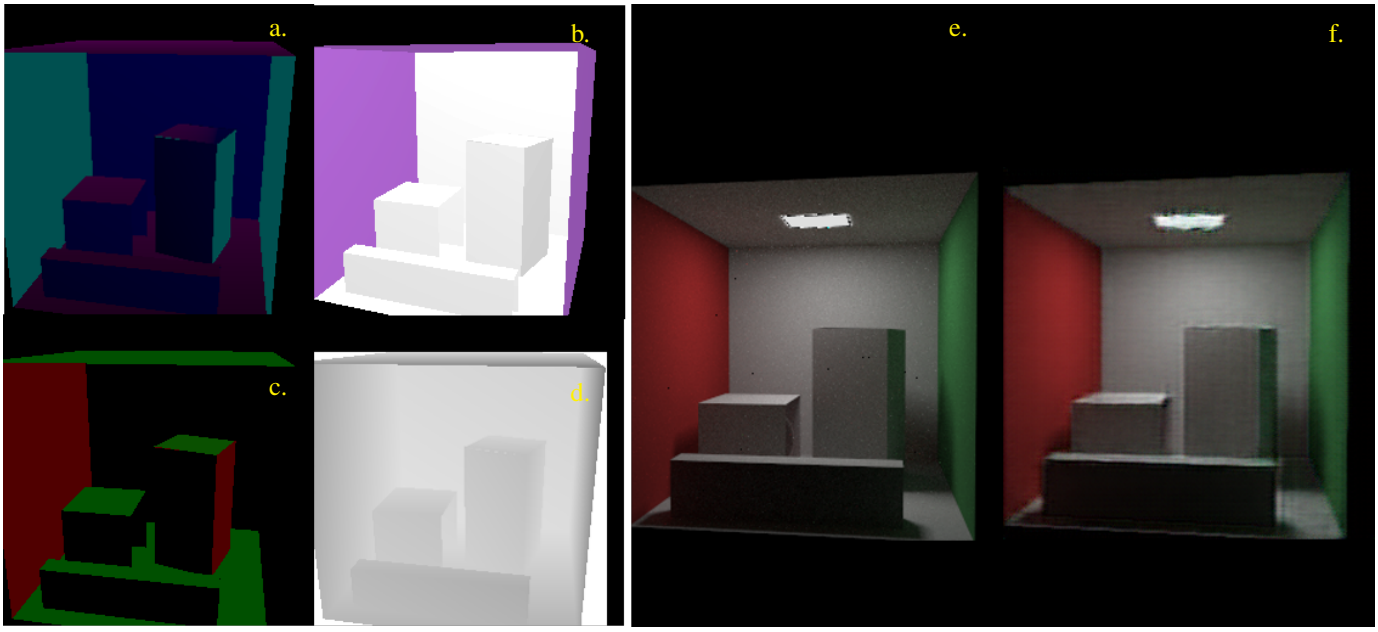


Fig. 2: Rendered Conditional Geometry Buffers, albedo, direct lighting, normals of surfaces and depth, ((a) – (d), respectively) and artificial rendering with conditional generative adversarial neural network (e–f) comparing ground truth path traced rendering (e) with image generated (f).

```
class PAVE
{
public:

    PAVE(std::string TSN);
    template<typename T>
    void save(std::string, int, int, std::vector<T>&);
};

constexpr int width = 256;
constexpr int height = 256;
std::vector<float> image(width*height);
PAVE paver("TrainingSetName");
paver.save(varName,
           width,
           height,
           image);
```

1: C++ Interface for PAVE

```
import PAVE
from torch.utils.data import DataLoader

def load_data(visualisation_vars=None, path = None):
    if path:
        train_set  = PAVE.DataLoader(path)
        train_data = DataLoader(dataset = train_set)
        return train_data
    else:
        PAVE.open(visualisation_vars)

def train(Model, simulation_variables):
    for (i, var) in enumerate(visualisation_vars):
        train_sample = load_data(var)
        prediction = Model(train_sample)
        ...
```

2: Python Interface

which will flush the current in-memory data to disk. A class for the C++ visualization interface using `std::vector` is in Listing 1.

B. User Defined Machine Learning Application

PAVE allows researchers or practitioners to implement their learning algorithms in the increasingly popular language Python due to having a robust library for learning tasks and notably neural networks.

In Listing 2, we demonstrate coupling PAVE while training a PyTorch model. PAVE's design pattern within Python as used for learning, or more generally for in situ or read retrieval, is through the use of `PAVE.DataLoader`, `open`, `read`, `close`. `PAVE.DataLoader` is a virtual function that is to be overloaded by the user and can be instantiated with either the name of the desired data or the parameter set needed for the visualization task defined by the user. With `open`, visualization output is read in situ from datum buffered by PAVE. With `read`, PAVE will retrieve written visualization information to be used for analysis. In the example provided the training method for "Model" is able to request visualisation samples to use for training by employing PAVE to iterate over data passed in situ or read.

C. Communication of Visualisation Data and Learning

As demonstrated in Section III-A and PAVE allows for the user to save or pass data produced by the visualization and similarly the user would also be able to request results from the learning model depending on the application. Section III-B demonstrates PAVE's support in requesting data in place from

the user's visualization implementation during training of a predictive model used as example.

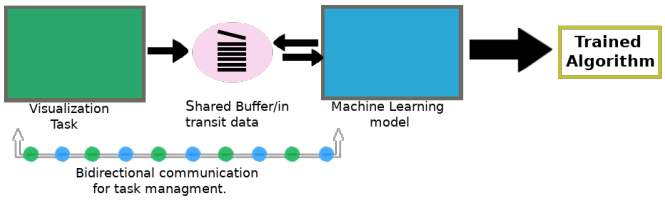


Fig. 3: A high level overview of the in situ data transport. Initially, a visualization task, such as global illumination, is generated and "staged" using PAVE. The machine learning model can then access the "staged" data and generate a trained algorithm from the visualization output.

The contribution of this work is then to provide a fluid and efficient workflow as demonstrated in Fig. 3. The machine learning methodology takes visualization output and data to build a ML trained algorithm (Fig. 1). The visualization output required can be significant in size. Therefore, it is "staged" utilizing PAVE, which then makes it accessible to the ML framework. Once the ML framework finally generates a trained algorithm, which can then take new visualization data and apply the new trained algorithm to it.

IV. Case Study: Coupled Path Tracing and Machine Learning

To demonstrate the generalized workflow afforded by PAVE of in situ, in transit, visualization, we provide a case study. This case study utilisation of PAVE will consist of three consecutive phases: rendering phase of conditional training images, training phase of the generative neural network, and execution phase of the trained network. Three core components, VTK-m, PyTorch, and PAVE, each fulfill a unique functional requirement during the separate stages. This case study also serves to demonstrate the support PAVE can provide in the context of deep learning. In this section we describe the independent design and global role each system plays.

A. System Overview

To achieve our goal of a conditional generative neural network (cGAN) capable of rendering geometric dependent object path simulations we begin by rendering ground truth scene images along with informative conditional image buffers. For this purpose VTK-m was chosen due to its scalability, heterogeneous backends, and robust capability for HPC visualisation tasks. Provided the training set of conditional and ground truth images two neural networks, one convolutional and one generative, play a zero-sum game common to training cGANs. To segue data management of training images the path tracer, saves the training set in a distributed setting with the use of PAVE. During training PyTorch is then able to retrieve needed image data through the use of the PAVE.

In this context we aim to demonstrate an application of PAVE's ability to provide support for deep learning models to train and interact in parallel with other visualization tasks. PAVE is then able to manage any differences that may arise in the often long training time of deep neural networks and that of the visualization objective while maintaining in situ transfer of data as well as accommodating the I/O demands required of each application in a scalable manner.

B. Path Tracer Design

High quality ground truth rendered scenes and conditional image attributes are required for the training stage. For this reason, the first stage of PAVE consists of generating a visual scene or simulation with VTK-m.

Within the parallel primitives framework of VTK-m, a path tracer was implemented to render high quality ground truth scenes. The path tracer utilizes Monte Carlo sampling through probability distribution functions over shapes of interest and light intensity and pixel values with cumulative distribution function sampling, light scattering, randomly directed light paths and material sampling. At a high level, the sampling is performed by sending multiple rays per pixel. The number of samples per pixel (SPP) is a user defined value. The ray continues to traverse the scene, accumulating the color of the surfaces it interacts with until it hits a light source. The accumulated color is normalized and stored to the image buffer. For a more in-depth examination of path tracing, see [22]. This procedure generates realistic lighting for a scene. To speed-up the scene rendering, a BVH, a tree-based acceleration structure, is constructed over the primitives in a similar manner to [24].

Further, the image buffers needed to compute light paths afford an informative conditional dependency on the behavior of lighting based on the geometry and light sources within a scene. The VTK-m ray tracer [24] was adapted to render these conditional buffers, namely albedo, direct lighting, normals of surfaces and depth with respect to point of view are then stored or passed to PyTorch with PAVE, allowing for a a pipeline which preserves the solutions scalability.

C. Neural Network Design

The cGAN used closely follows that introduced by Thomas and Forbes with Deep Illumination [21]. Both the discriminator and generator network are deep convolutional neural networks implemented in PyTorch using training data retrieved from PAVE training sets formatted and stored by the VTK-m path tracer. The training stage relies on four conditional buffers depth, albedo, normals and direct lighting along with an associated ground truth image of high light sample count and ray depth.

Given the four conditional buffers the generator attempts to construct the ground truth image from noise. The discriminator is then fed both the generated and ground truth image. The loss used for the gradient back propagation update of both networks is based on the quality of the discriminators ability to classify the artificial and true image in which the generator is greater

penalised when the discriminator accurately differentiates the two images, and similarly, the discriminator has a larger loss when incorrectly identifying real from fabricated images. The generator is then considered to have converged when the discriminator predicts both generated and true images with equal probability. For both discriminator and generator networks the activation functions used between layers is LeakyReLu and Sigmoid for the final layer [25]. Batch normalisation is also performed between internal layers to minimise covariant shift of weight updates and improve learning for the deeper networks used [26].

1) Generator Network: The generative network used is a deep convolutional network consisting of an encoder and decoder with skip connections-concatenations of equal depth layers within the encoding and decoding stages. Due to the illustrative 'shape' of this design the network is denoted a U-Net as introduced by Ronneberger et. al. for medical segmentation [27]. The motivation for utilising a U-Net is due to success of the skip connections in capturing geometric and spatial attributes by linking the decoded convolutional process to the encoded upconvolutional. The mapping of contracting feature segmentation onto expanding upsampling within the network allows us to also exploit nearness to object geometry within the constructed and target image through Euclidean distance. As a result, performance in terms of required training time, quality of preserved structure, and accuracy maintaining light information of generated images drastically improves with the addition of an L1 loss to the classic binary cross entropy common to training adversarial networks when updating the discriminator and generator [28] [29].

2) Discriminator Network: For discriminating between artificial and ground truth image renderings a deep convolutional patchGAN network is used motivated by the added advantage of providing a patch-wise probability of an image in question as being real or fake. The benefit of a patch-wise probability allows for higher regional accuracy within an image as well as applicable for image-to-image tasks as introduced by Isola et. al. [30]. The image classification probability is then interpreted as the average of these patch wise probabilities over an image in question.

As input during training the discriminator network is given the set of conditional space buffers along with either the visualisation generated by the cGAN generator network or the ground truth global illumination rendering produced with the VTK-m path tracer. Taking into account the conditional image buffers stacked atop an image sample an image sample under question the resulting input is a tensor of the form Width x Height x 15. The discriminator then attempts to predict with what probability the provided image stack, either fabricated by the U-Net or the VTK-m path tracer, is real. Based on the discriminators performance the loss is computed using the classic loss for training GANs along with an L1 loss. Training is complete, and the generator has converged, when the discriminator predicts images as real or fake with an equal probability, e.g. 50% chance of accuracy.

At this point, once training on the target visualization has completed, the generator can be considered capable of emulating it's scene dependent objective. The discriminator network is then discarded and the resulting generator affords a scene dependent real-time *in situ* visualisation tool able to produce accurate global illumination from conditional geometric buffers.

D. Our Provided Example of PAVE

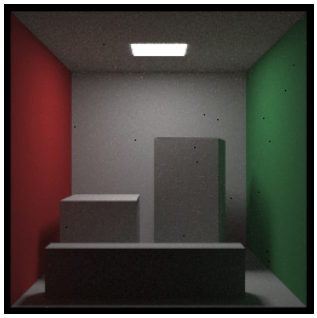
For our PyTorch *in situ* proposal the current systems we employ are VTK-m, for rendering path traced images and conditional geometry buffers focus, and PAVE for data management. We discuss the design pattern for our *in situ* visualisation task with support from deep learning in the order of operations followed within the pipeline of use. Namely, we present the design pattern for rendering light transport in VTK-m coupled with data transport to PyTorch (IV-D1). Subsequently we explain the infrastructure for embedding VTK-m throughput managed by PAVE within PyTorch and demonstrate through example (IV-D2) by instantiating a PyTorch data interface allowing for data parallelization and multi-GPU/distributed training as used within the framework for training our cGAN model.

1) VTK-m Light Transport Visualisation: Path traced images are maintained within C++11 as VTK-m arrays which can be passed by reference directly to PyTorch using PAVE and retrieved during training or when utilising the neural network to generate novel scene renderings from rendered geometry buffers.

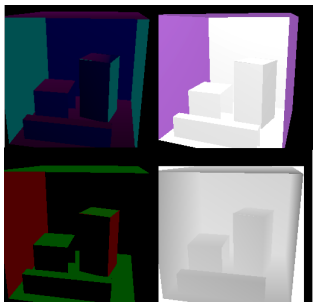
2) PyTorch cGAN Global Illumination Generator: For training, our solution used by the cGAN is "*PAVEDataLoader*", a data class inheriting from the abstract indexing class PyTorch *torch.utils.data.Dataset*. The *PAVEDataLoader* employs PAVE to retrieve representations of path traced images and conditional buffers. Within VTK-m during generation these array are represented as VTK-m Arrays and within PyTorch as numpy arrays. In this manner the training or test sets needed by PyTorch and created by VTK-m are available to PyTorch *in situ*. VTK-m generated datasets can be retrieved with *read()* or passed to a similar *get()* and subsequently forwarded to our *get_split()* to partition the dataset into 60% training, 20% testing and 20% validation subsets. The split datasets are then used to construct the *PAVEDataLoader* class which inherits from the *torch.utils.data.DataLoader* thereby providing a data sampler of our VTK-m renderings with a single-process or multi-process iterator over the dataset, providing the tools necessary to train our neural networks in the canonical manner.

V. Cornell Box Experiment

To asses *in situ* deep learning aided visualisation we train the cGAN network on rendered images of a variant of the Cornell box [31], a commonly used 3D model for quality assessment. We train the model using renderings of the Cornell box with high light sample count and depth computation per ray for various camera angle perspectives directed into the box along with the associated image geometry buffers for a given



(a) Path traced global illumination rendered



(b) Geometry buffers

Fig. 4: (a) Ground truth image rendered with VTK-m used for training. (b) Global illumination geometry buffers used as conditional variables for generative model. **Top:** left, Albedo. right, Direct Lighting. **Bottom:** left, Normals. right, Depth.

camera orientation. We then assess the quality of the models final generated renderings looking at the accuracy of global illumination. The scene used for training is comprised of the classic set up with one overhead light source in the center of a white ceiling, a white back wall and a white floor. The remaining walls are then colored red on the left and green on the right in order to afford different colored light transport and demonstrate diffuse interreflection. The contents of the Cornell box are three cuboids of various shapes and sizes to provide diverse shading and diffused lighting.

The conditional differed shading geometry buffers used are direct lighting, normal planes, depth and albedo as shown in figure 2.

The geometry buffers serve as joint variables for the conditional probability distribution which the global illumination path traced images are considered to exist. The conditional arguments in this experiment then aid the cGAN in learning behavior of light paths given the geometry of a scene in question.

VI. Results

A. Cornell Box Experiment Results

A total of 3080 high quality images of dimensions 256x256x3 were used as the ground truth data set and were rendered utilizing the VTK-m path tracer with a ray depth cut-off of 50 and 1000 samples per pixel, and two Nvidia RTX-2080 Ti GPUs requiring a total of 13 hours. The 3080 training images were rendered by rotating the camera position around a hemisphere while maintaining the camera's focal point at the center of the Cornell Box. Each image was considered a single datum (Sec. III-A) in the path traced training set given to the cGAN. The cGAN was trained on this image data set for 400 epochs using one GPU on the same machine took 9709 seconds.

The resulting generated images show promising results for deep learning aided *in situ* scientific visualisation. We observe the network successfully learned to emulate light transport in a realistic fashion similar in quality to the offline performance, as can be seen in the highlighted region of Fig. 5. It should be noted, the images of Fig. 5 are taken from the partitioned test set. The examples provided were then not utilized during training and were unseen by the network until test time. The ground truth image is then shown for comparison. Further, though designed in a "one network for one scene" setting, the generative net can be used to render scenes from camera orientations not in the training set, but still deliver approximate physically-based lighting.

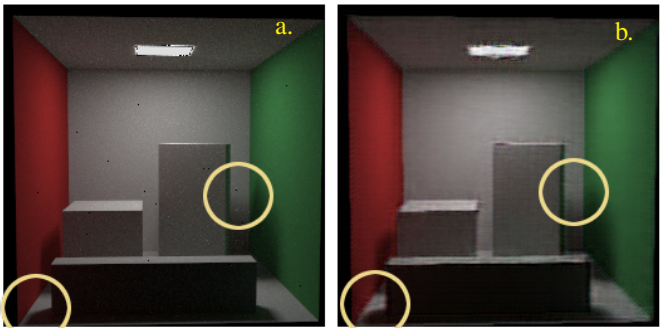


Fig. 5: Ground truth path traced image (a) and resulting image generated by cGAN from noise and test sample conditional buffers excluded from training (b).

Observing the respective loss plots of the generative network and discriminator network during training demonstrates the success of the generative network due to the discriminator's loss converging near 0.5. With loss initially near zero, the discriminator was able to easily differentiate the noise produced by the generator from the ground truth images. As training progressed, and the generative network improved with loss converging to zero, the discriminator deteriorated and was left with even odds amounting to a near 50-50% chance of identifying input images as either artificial or original.

VII. Conclusions

We have presented an in situ framework to couple scientific visualization applications to machine learning applications. Our work offers a scalable framework to enable researchers and practitioners to integrate state of the art deep learning tools for scientific visualization. We presented a case study to demonstrate the ease of integration with PyTorch and robust visualisation resource VTK-m for in situ visualization for HPC systems. The case study utilises cGANs for generative visualisation which also offers the prospect of quickly and accurately visualising conditionally dependent data such as light path global illumination's dependence on scene geometry.

In the future, we would like to scale beyond two GPUs to examine the trade offs in performance between the visualization and machine learning tasks and importantly, using in situ in-transit visualization on a large scale system. Further, we would like to integrate other visualization tools, such as Intel's OSPRay or Nvidia's OptiX, as well as, other machine learning frameworks such as TensorFlow or Caffe to examine

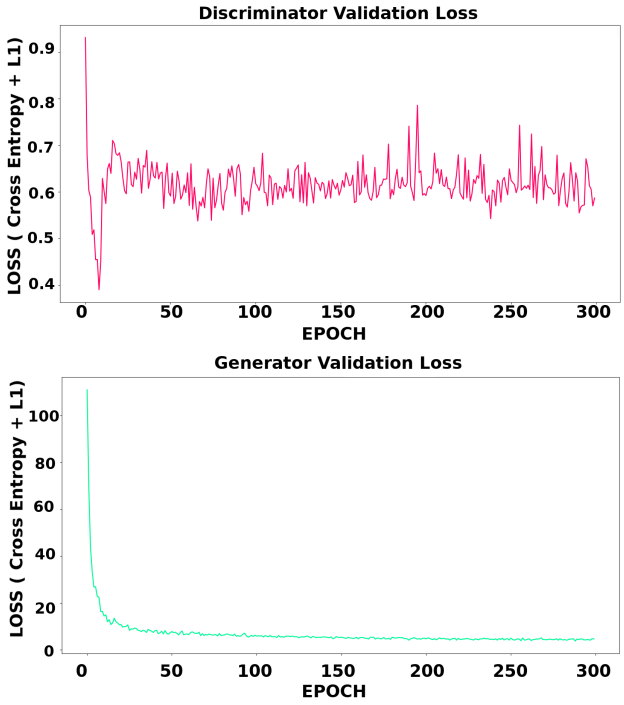


Fig. 6: Discriminator loss converging to 0.5 and generator loss converging to 0 corresponding to inability to differentiate real from generated images.

trade-offs in performance and quality as well. Finally, we would like to validate whether machine learning filters for global illumination are of sufficient fidelity to use for scientific visualization applications in HPC.

Acknowledgment

This research was supported in part by an appointment to the Oak Ridge National Laboratory ASTRO Program, sponsored by the U.S. Department of Energy and administered by the Oak Ridge Institute for Science and Education and is supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

References

[1] K. Moreland, "The tensions of in situ visualization," *IEEE Computer Graphics and Applications*, vol. 36, no. 2, pp. 5–9, Mar 2016.

[2] A. C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. Moreland, P. O'Leary, V. Vishwanath, B. Whitlock, and E. W. Bethel, "In situ methods, infrastructures, and applications on high performance computing platforms," *Computer Graphics Forum*, vol. 35, no. 3, pp. 577–597, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12930

[3] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng, "Datastager: scalable data staging services for petascale applications," *Cluster Computing*, vol. 13, no. 3, pp. 277–290, Sep 2010. [Online]. Available: https://doi.org/10.1007/s10586-010-0135-6

[4] U. Ayachit, A. Bauer, B. Geveci, P. O'Leary, K. Moreland, N. Fabian, and J. Mauldin, "Paraview catalyst: Enabling in situ data analysis and visualization," in *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ser. ISAV2015. New York, NY, USA: ACM, 2015, pp. 25–29. [Online]. Available: http://doi.acm.org/10.1145/2828612.2828624

[5] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. Favre, and P. Navratil, "VisIt: An End-User Tool for Visualizing and Analyzing Very Large Data," in *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, ser. Chapman & Hall, CRC Computational Science, E. W. Bethel, H. Childs, and C. Hansen, Eds. Boca Raton, FL, USA: CRC Press/Francis–Taylor Group, Nov. 2012, pp. 357–372, http://www.crcpress.com/product/isbn/9781439875728, LBNL-6320E.

[6] J. Dayal, D. Bratcher, G. Eisenhauer, K. Schwan, M. Wolf, X. Zhang, H. Abbasi, S. Klasky, and N. Podhorszki, "Flexpath: Type-based publish/subscribe system for large-scale science analytics," in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2014, pp. 246–255.

[7] U. Ayachit, B. Whitlock, M. Wolf, B. Loring, B. Geveci, D. Lonie, and E. W. Bethel, "The sensei generic in situ interface," in *Proceedings of the 2Nd Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization*, ser. ISAV '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 40–44. [Online]. Available: https://doi.org/10.1109/ISAV.2016.13

[8] M. Larsen, J. Ahrens, U. Ayachit, E. Brugger, H. Childs, B. Geveci, and C. Harrison, "The alpine in situ infrastructure: Ascending from the ashes of strawman," in *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization*, ser. ISAV'17. New York, NY, USA: ACM, 2017, pp. 42–46. [Online]. Available: http://doi.acm.org/10.1145/3144769.3144778

[9] M. Larsen, A. Woods, N. Marsaglia, A. Biswas, S. Dutta, C. Harrison, and H. Childs, "A flexible system for in situ triggers," in *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ser. ISAV '18. New York, NY, USA: ACM, 2018, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/3281464.3281468

[10] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, M. Parashar, N. Samatova, K. Schwan, A. Shoshani, M. Wolf, K. Wu, and W. Yu, "Hello adios: the challenges and lessons of developing leadership class i/o frameworks," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 7, pp. 1453–1473, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3125

[11] J. Kress, M. Larsen, J. Choi, M. Kim, M. Wolf, N. Podhorszki, S. Klasky, H. Childs, and D. Pugmire, "Comparing the Efficiency of In Situ Visualization Paradigms at Scale," in *ISC High Performance*, Frankfurt, Germany, Jun. 2019, pp. 99–117.

[12] NVIDIA Corporation, *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation, 2007.

[13] J. Reinders, *Intel threading building blocks - outfitting C++ for multi-core processor parallelism*. O'Reilly, 2007.

[14] J. Hoberock and N. Bell, "Thrust: A parallel template library," *Thrust: A Parallel Template Library*, 2009.

[15] G. E. Blelloch, *Vector Models for Data-parallel Computing*. Cambridge, MA, USA: MIT Press, 1990.

[16] K. Moreland, C. Sewell, W. Usher, L.-t. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K.-L. Ma, H. Childs *et al.*, "Vtk-m: Accelerating the visualization toolkit for massively threaded architectures," *IEEE computer graphics and applications*, vol. 36, no. 3, pp. 48–58, 2016.

[17] J. T. Kajiya, "The rendering equation," in *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '86. New York, NY, USA: ACM, 1986, pp. 143–150. [Online]. Available: http://doi.acm.org/10.1145/15922.15902

[18] A. Keller, L. Fascione, M. Fajardo, I. Georgiev, P. Christensen, J. Hanika, C. Eisenacher, and G. Nichols, "The path tracing revolution in the movie industry," in *ACM SIGGRAPH 2015 Courses*, ser. SIGGRAPH '15. New York, NY, USA: ACM, 2015, pp. 24:1–24:7. [Online]. Available: http://doi.acm.org/10.1145/2776880.2792699

[19] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," in *ACM Transactions on Graphics (TOG)*, vol. 21, no. 3. ACM, 2002, pp. 527–536.

[20] A. Kaplanyan and C. Dachsbacher, "Cascaded light propagation volumes for real-time indirect illumination," in *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM, 2010, pp. 99–107.

[21] M. M. Thomas and A. G. Forbes, "Deep illumination: Approximating dynamic global illumination with generative adversarial network," *arXiv preprint arXiv:1710.09834*, 2017.

[22] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.

[23] M. Zwicker, W. Jarosz, J. Lehtinen, B. Moon, R. Ramamoorthi, F. Rousselle, P. Sen, C. Soler, and S.-E. Yoon, "Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering," *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports)*, vol. 34, no. 2, p. 667681, May 2015.

[24] M. Larsen, J. Meredith, P. Navratil, and H. Childs, "Ray tracing within a data parallel framework," in *2015 IEEE Pacific Visualization Symposium, PacificVis 2015 - Proceedings*, ser. IEEE Pacific Visualization Symposium, S. Takahashi, S. Liu, and G. Scheuermann, Eds. United States: IEEE Computer Society, 7 2015, pp. 279–286.

[25] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[26] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[27] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[28] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[29] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[30] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[31] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile, "Modeling the interaction of light between diffuse surfaces," in *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '84. New York, NY, USA: ACM, 1984, pp. 213–222. [Online]. Available: http://doi.acm.org/10.1145/800031.808601