PAVE

Samuel Leventhal*
samlev@cs.utah.edu
University of Utah School of Computing: Scientific Computing and Imaging Institute

Mark Kim
kimmb@ornl.gov
Oak Ridge National Lab

Dave Pugmire
pugmire@ornl.gov
Oak Ridge National Lab

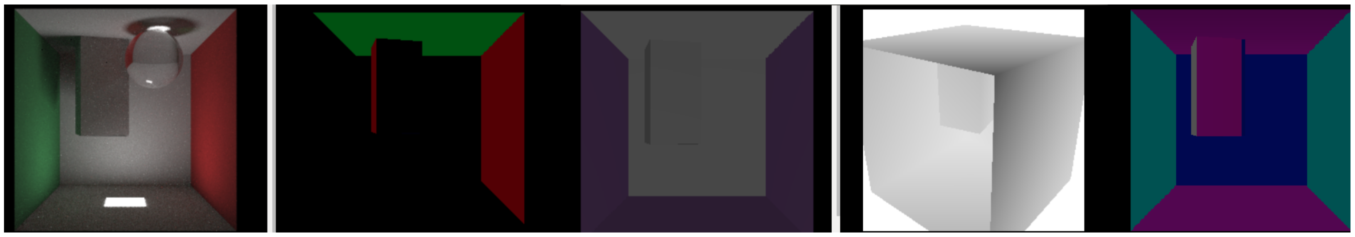


Figure 1: Rendered Conditional Images

ABSTRACT

In this work we offer an approachable platform for visualization tasks by employing a neural network for real time rendering and accurate light transport simulation within the framework of Python made compatible for distributed systems and high performance computing (HPC). The provided model is a coalescence of VTK-m, a visualization tookit fit for massively threaded architectures, PyTorch, an increasingly popular language within machine learning due to robust libraries for neural networks, and Adios, an adaptable unified IO framework for data managment at scale. The resulting work accomplishes this combination by utilizing VTK-m to construct a path trace renderer able to fluidly and efficiently communicate to a conditional Generative Advisarial Network (cGAN) by means of Adios during training. The resulting generative model serves as a filter for rendered images and visual simulations capable of approximating indirect illumination and soft shadows at real-time rates while maintaining quality comparable to offline approaches. "in situ deep learning for scientific visualization"

CCS CONCEPTS

• **Theory of computation → Parallel computing models**; **Distributed computing models**; **Structured prediction**; **Adversarial learning**; **Data structures and algorithms for data management**; *Probabilistic computation*; *Database query languages (principles)*; • **Applied computing → Computer-aided design**.

KEYWORDS

VTKm, neural networks, generative adversarial network, Adios, PyTorch, path tracing

ACM Reference Format:
Samuel Leventhal, Mark Kim, and Dave Pugmire. 2019. PAVE. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 APPLICABLE "AREA OF INTERESTS" TARGETS

(1) In situ data management and infrastructures Current Systems: production quality, research prototypes , Opportunities ,Gaps
Current Systems: integration of VTKm, Adios2 and Python (PyTorch). Prototype being a conditional generative adversarial network (cGAN) designed to use a VTKm based pathtracer applied but not limited to learning global illumination and light behavior in rendering tasks. Opportunities: Introducing a framework allowing researchers easy access to python on HPC systems as well as machine learning aided technique to treat and study experimental data used in scientific simulations as learnable probability distributions with derived conditional dependencies of interest.
(2) System resources, hardware, and emerging architectures. Enabling Hardware, Hardware and architectures that provide opportunities for In situ processing, such as burst buffers, staging computations on I/O nodes, sharing cores within a node for both simulation and in situ processing
Enabling Hardware: By constructing an architecture allowing for Python to interface with VTKm data management controlled by Adios2 the proposed software

allows for a well distributed simulation task among cores.
(3) Methods and algorithms: Analysis: feature detection, statistical methods, temporal methods, geometric and topological methods Visualization: information visualization, scientific visualization, time-varying methods
(4) Case Studies and Data Sources In situ methods/systems applied to data from simulations and/or experiments / observations
(5) Simulation and Workflows: Integration:data modeling, software-engineering, Workflows for supporting complex in situ processing pipelines
(6) Requirements, Usability: Reproducibility, provenance and metadata

2 INTRODUCTION

3 RELATED WORK

Real time true to life quality renerings of light transport remains an active area of research with a number of various approaches. 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 [11]. Similarly, Light Propagation Volumes have been used to iteratively propagate light between consecutive grid positions to emulate single-bounce indirect illumination [6]. 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 advisarial 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 [12].

GAN [2]
cGAN [8]
Tomas and Forbes Deep Illumination:
VTKm [9]
Reinforced learning for light transport simulation [1]

4 TECHNIQUE OVERVIEW

Utilization of PAVE consists 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 Adios2 fufill a unique functional requirement during each stage. In this section we describe the independent design and global role each system plays.

4.1 System Overview

To achieve our goal of a conditional generative neural network capable of rendering geometric dependent object path simulations we begin by rendering informative conditional image buffers along with ground truth scene renderings. For this purpose the VTK-m was chosen due to its scalability and robust capability for HPC visualization 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 GANs. To segue data managment of training images the path tracer saves the training set in a distributed setting with the use of Adios2. During training PyTorch is then able to retrieve needed image data through the use of the adaptable IO provided by Adios's Python high-level APIs.

4.2 Path Tracer Design

Conditional image attributes and high quality ground truth rendered images 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 framework of VTK-m the implemented ray tracer renders images through means common to commercial ray tracers such as Monte Carlo sampling for shapes of interest, light scattering, randomly directed light paths, material sampling and direct sampling. The image buffers needed to compute light paths afford an informative conditional dependence on the behavior of lighting based on the geometry and light sources within a scene. These conditional buffers, namely albedo, direct lighting, normals of surfaces and depth with respect to camera are then stored within VTK-m with Adios to maintane scalability of the system. For subsequent phases of PAVE the training data can then be retrieved from file again through Adios differing only in the API needed.

4.3 Neural Network Design

The cGAN used closely follows that introduced by Thomas and Forbes with Deep Illumunation [12]. Both the desriminator and generator network are deep convolutional neural networks implemented in PyTorch using training data retrieved from Adios files formated 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 backpropagation update of both networks is based on the quality of the descriminators ability to classify the artifical and true image in which the generator is greater penalized 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 descriminator predicts both generated and true images with equal probability. For both descriminator and generator networks the activation functions used between layers is LeakyReLu and Sigmoid for the final layer [7]. Batch normalization is also performed between internal layers to minimize covariant shift of weight updates and improve learning for the deeper networks used [3].

4.3.1 Descriminator Network. For descriminating 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. [4].

As input during training the descriminator network is given the set of conditional space buffers along with either the visualization 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 buffers the descriminator attempts to provide the rendered image as artifical, e.g. generated by the advisarial network, or real. Based on the performance of the descriminator the loss is computed using the classic loss for GAN training along with an L1 loss in order to not only produce original content but to also preserve structure and light information [5]. Training is complete, and the generator has converged, when the discriminator predicts real from fake images with a 50-50 percent chance. At this point the desciminator network is discarded and the resulting generator affords a real-time visualization tool able to produce accurate global illumination given conditional geomretic buffers.

4.3.2 Generator Network. The generative network used is a deep convolutional network consisting of an encoder and decoder with skip connections concatonating equal depth layers of 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 [10]. The motivation for utilizing a U-Net is due to success of the skip connections linking the decoded convolutional process to the encoded deconvolutional in capturing geometric and spatial attributes. The generator retreives as input through Adios global illumination buffers saved to file once rendered with VTK-m.

4.4 Core Design Pattern

For our PyTorch in situ proposal our means for generating path traced images and geometry buffters focus on VTK-m and are succesfuly incorporated into PyTorch with Adios2. Our main focus is Adios2's IO API from within C++, utilizing write capabilities, and Python oriented towards data retreival.

4.4.1 VTK-m Data Generation and Adios2 Write.

4.4.2 PyTorch with Adios2 IO. For training the cGAN our solution provides the AdiosDataLoader to be used by PyTorch during training.

```
import read_adios_bp
import get_split
class AdiosDataLoader(data.Dataset):
    #split can be 'train', 'val', and 'test'
    def __init__(self, image_dir, split = 'train'):
        super(AdiosDataLoader, self).__init__()
        self.albedo_path = join(image_dir, "albedo.bp")
        self.depth_path = join(image_dir, "depth.bp")
        self.direct_path = join(image_dir, "direct.bp")
        self.normal_path = join(image_dir, "normals.bp")
        self.gt_path = join(image_dir, "outputs.bp")
        self.width = 256
        self.height = 256
        self.samplecount = 50
        # albedo
        self.image_data = read_adios_bp(self.gt_path
                    ,conditional="trace"
                    ,width=self.width
                    ,height=self.height
                    ,sample_count=self.samplecount)

        #collect adios var (image) names
        self.image_filenames = self.image_data[0]

        # partition training, testing and validation set
        self.split_filenames = get_split(self.image_filenames,
                                         split)
        # albedo image name to image map
        self.name_to_albedo = read_adios_bp(self.albedo_path
                        ,conditional="albedo"
                        ,width=self.width
                        ,height=self.height
                        ,sample_count=self.samplecount

        #depth image name to image dict
        self.name_to_depth = read_adios_bp(self.depth_path
                        ,conditional="depth"
                        ,width=self.width
                        ,height=self.height
                        ,sample_count=self.samplecount
        # direct  image name to image dict
        self.name_to_direct = read_adios_bp(self.direct_path
                        ,conditional="direct"
                        ,width=self.width
                        ,height=self.height
                        ,sample_count=self.samplecount
        # path traced image image name to image dict
        self.name_to_trace = read_adios_bp(self.gt_path
                        ,conditional="trace"
                        ,width=self.width
                        ,height=self.height
                        ,sample_count=self.samplecount
        # normal buffer image name to image dict
        self.name_to_normals = read_adios_bp(self.normal_path
                        ,conditional="normals"
                        ,width=self.width
                        ,height=self.height
                        ,sample_count=self.samplecount
```

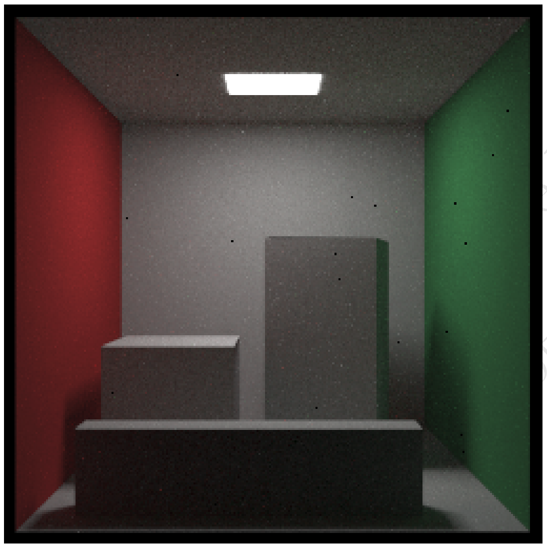
The *AdiosDataLoader* employs Adios2 to retrieve pre-generated path traced images and conditional buffers to be instantiated as a PyTorch *DataLoader*. Given a conditional image set of interest and path to the saved Adios .bp files a dictionary associating name to the accompanied RGBA image is retrieved with *read_adios_bp()* and is subsequently partioned into training, testing and validation subsets with the *get_split()* method into 60% training images, 20% validation images and 20% testing images.

Using the *AdiosDataLoader* we can then train our neural networks implemented in PyTorch in the canonical manner

```
#PyTorch Imports
from torch.utils.data import DataLoader
from torch.autograd import Variable
#Our model imports
from model import G, D
from data import AdiosDataLoader

opt = parser.parse_args()

train_set = AdiosDataLoader(opt.dataset,split="train")
val_set = AdiosDataLoader(opt.dataset, split="val")
test_set = AdiosDataLoader(opt.dataset, split="test")

batch_size = opt.train_batch_size
n_epoch = opt.n_epoch

# instantiate PyTorch dataloaders with Adios2DataLoader
train_data = DataLoader(dataset=train_set,
                        num_workers=opt.workers,
                        batch_size=opt.train_batch_size,
                        shuffle=True)
val_data = DataLoader(dataset=val_set,
                      num_workers=opt.workers,
                      batch_size=opt.test_batch_size,
                      shuffle=False)
test_data = DataLoader(dataset=test_set,
                       num_workers=opt.workers,
                       batch_size=opt.test_batch_size,
                       shuffle=False)

albedo = torch.FloatTensor(opt.train_batch_size,
                           opt.n_channel_input,
                           256, 256)
direct = torch.FloatTensor(opt.train_batch_size,
                           opt.n_channel_input,
                           256, 256)
normal = torch.FloatTensor(opt.train_batch_size,
                           opt.n_channel_input,
                           256, 256)
depth = torch.FloatTensor(opt.train_batch_size,
                          opt.n_channel_input,
                          256, 256)
gt = torch.FloatTensor(opt.train_batch_size,
                       opt.n_channel_output,
                       256, 256)
label = torch.FloatTensor(opt.train_batch_size)

netG = G(opt.n_channel_input*4,
         opt.n_channel_output,
         opt.n_generator_filters)
netD = D(opt.n_channel_input*4,
         opt.n_channel_output,
         opt.n_discriminator_filters)
netD = netD.cuda()
netG = netG.cuda()

# loss functions used
criterion = nn.BCELoss()
criterion_l1 = nn.L1Loss()

#cuda placement and instantiate
#PyTorch variables
criterion = criterion.cuda()
criterion_l1 = criterion_l1.cuda()

albedo = albedo.cuda()
direct = direct.cuda()
normal = normal.cuda()
depth = depth.cuda()
gt = gt.cuda()
label = label.cuda()

albedo = Variable(albedo)
direct = Variable(direct)
normal = Variable(normal)
depth = Variable(depth)
gt = Variable(gt)
label = Variable(label)

# instantiate PyTorch Adam gradiant decent
optimizerD = optim.Adam(netD.parameters(),
                        lr=opt.lr,
                        betas=(opt.beta1, 0.999))
optimizerG = optim.Adam(netG.parameters(),
                        lr=opt.lr,
                        betas=(opt.beta1, 0.999))

def train(epoch):
    for (i, images) in enumerate(train_data):
        netD.zero_grad()
        ...
```

5 CORNELL BOX EXPERIMENT

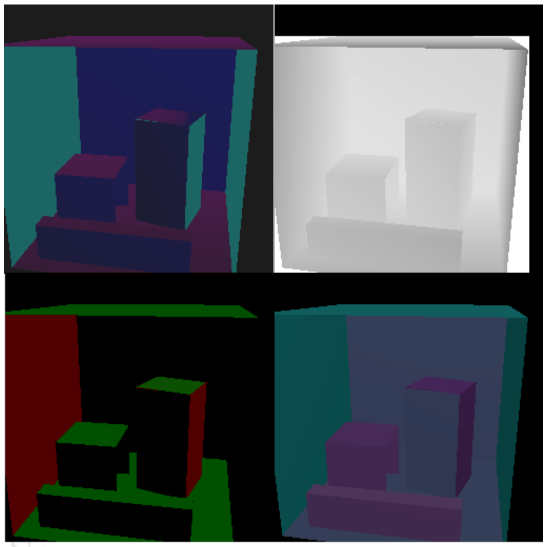
To evaluate the quality of in situ deep learning aided visualizations train the cGAN networks on rendered images of a Cornell box, a commonly used 3D modeling framework for quality assesment. We train the model using renderings of the Cornell box with high light sample count and depth computation per ray for various camera angle perspectives into the box along with the associated image geomretry buffers for a given camera orientation. We then assess the quality of the mdoels final generated renderings looking at the accuracy of global illumination. We then also demonstrate the performance of the models ability to render global illuminatio when given image buffers for a novel scene not used for training similar in content but not exact. 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 amd demonstrate diffuse interreflection. The contents of the Cornell box are three cuboids of various shapes and sizes to provide diverse shading and diffused lighting.



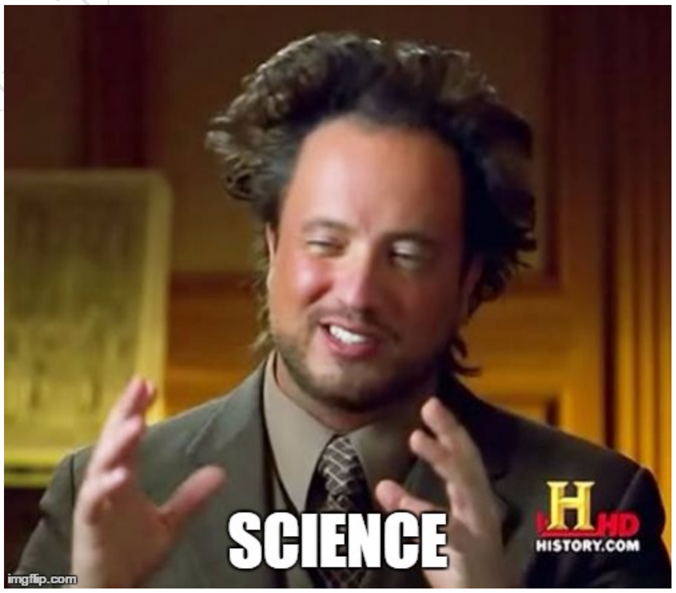
The conditional differed shading geometry buffers used are direct lighting, normal planes, depth and albedo.

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 behaviour of light paths given the geometry of a scene in question.

Figure 2: Global illumination conditional image buffers. **Top:** Albedo, left. Depth, right. **Bottom:** Normals, left. Direct lighting, right.



6 RESULTS



7 CONCLUSIONS

ACKNOWLEDGMENTS

Identification of funding sources and other support, and thanks to individuals and groups that assisted in the research and the preparation of the work should be included in an acknowledgment section, which is placed just before the reference section in your document.

REFERENCES

[1] Ken Dahm and Alexander Keller. 2017. Learning light transport the reinforced way. *arXiv preprint arXiv:1701.07403* (2017).

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. Advances in Neural Information Processing Systems. (2014).

[3] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[4] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1125–1134.

[5] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1125–1134.

[6] Anton Kaplanyan and Carsten Dachsbacher. 2010. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM, 99–107.

[7] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, Vol. 30. 3.

[8] Mehdi Mirza and Simon Osindero. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784* (2014).

[9] Kenneth Moreland, Christopher Sewell, William Usher, Li-ta Lo, Jeremy Meredith, David Pugmire, James Kress, Hendrik Schroots, Kwan-Liu Ma, Hank Childs, et al. 2016. Vtk-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE computer graphics and applications* 36, 3 (2016), 48–58.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.

[11] Peter-Pike Sloan, Jan Kautz, and John Snyder. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *ACM Transactions on Graphics (TOG)*, Vol. 21. ACM, 527–536.

[12] Manu Mathew Thomas and Angus G Forbes. 2017. Deep Illumination: Approximating Dynamic Global Illumination with Generative Adversarial Network. *arXiv preprint arXiv:1710.09834* (2017).

A APPENDIX