ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL05

RemoteVis: AN EFFICIENT LIBRARY FOR REMOTE VISUALIZATION OF LARGE VOLUMES USING NVIDIA INDEX

T. V. Spina*, D. A. D. Alnajjar, M. L. Bernardi, F. S. Furusato, E. X. Miqueles, A. Z. Peixinho
Brazilian Synchrotron Light Laboratory, CNPEM, Campinas, Brazil
A. Kuhn, M. Nienhaus, NVIDIA, Berlin, Germany

Abstract

Advancements in X-ray detector technology are increasing the amount of volumetric data available for material analysis in synchrotron light sources. Such developments are driving the creation of novel solutions to visualize large datasets both during and after image acquisition. Towards this end, we have devised a library called RemoteVis to allow the visualization of large volumes remotely in HPC nodes, using NVIDIA IndeX as the rendering backend. RemoteVis relies on RDMA-based data transfer to move large volumes from local HPC servers, possibly connected to X-ray detectors, to remote dedicated nodes containing multiple GPUs for distributed volume rendering. RemoteVis then injects the transferred data into IndeX for rendering. IndeX is a scalable software capable of using multiple nodes and GPUs to render large volumes in full resolution. As such, we have coupled RemoteVis with slurm to dynamically schedule one or multiple HPC nodes to render any given dataset. Remote-Vis was written in C/C++ and Python, providing an efficient API that requires only two functions to 1) start remote IndeX instances and 2) render regular volumes and point-cloud (diffraction) backend. In-memory data is transferred directly to dedicated servers over the network via *Remote Direct Memory Access* (RDMA), without involving temporary file transfers or centralized storage. NVIDIA IndeX is a scalable software for interactive visualization of large volumes in full resolution, written in CUDA and designed to render data leveraging multiple GPUs and/or multiple nodes of a distributed HPC environment. When the volume is received by an instance of IndeX, it is immediately injected for visualization by the user, who can interact with the volume in real time using a web viewer.

INTRODUCTION

Improvements in synchrotron light source technology are pushing forward the boundaries of X-ray microscopy imaging. Recently, 4th generation synchrotron light sources are increasing the amount of available beam flux and coherence, opening the doors to novel imaging techniques while representing an improvement of orders of magnitude with respect to previous generations. Hence, the entire imaging pipeline is evolving to make those powerful improvements available to the beamlines and their users; starting with the creation of fast X-ray detectors capable of acquiring frames at multiple kHz and reaching the development of high performance data processing, visualization, and analysis workflows.

In this paper, we propose a volumetric data visualization workflow in the form of a library called RemoteVis, to address some of the challenges imposed by the large data volumes being generated. RemoteVis is designed as an efficient C/C++ API for sending image volumes for remote 3D rendering, using NVIDIA IndeX [1] as the rendering

The combination of RemoteVis and NVIDIA IndeX aims to overcome several limitations of existing open source and commercial visualization softwares. For instance, Neuroglancer [2] is a community-supported tool for visualization of very large volumes originally created by Google. It is capable of displaying arbitrary (non axis-aligned) cross-sectional views of volumetric data, as well as 3D meshes and line-segment based models (skeletons). Neuroglancer uses a tiling mechanism to handle zooming with different resolutions of large volumes, which are displayed on the web browser. Napari [3] allows simplified visualization of n-D data via Python, while ImageJ/Fiji contains plugins for rendering volumes [4]. 3D Slicer [5] is a rich application devoted primarily to medical imaging, containing several visualization tools for these types of data.

Despite some of the advantages of the aforementioned softwares, they either make limited use of GPU capabilities for rendering 3D volumes, relying on a single device to do so, or only displaying cross-sections of data, while providing less than optimal techniques to handle larger volumes (i.e., data sets with more than 2048^3 voxels). Even commercial solutions, such as ORS DragonFly [6] and Thermo Fisher OpenInventor [7] (Avizo/Amira), tend focus on single GPU/single node rendering, with limited APIs that can be used to address the needs of modern synchrotron light sources. Finally, those softwares are usually implemented considering that data is essentially stored on disk.

We initially designed RemoteVis to tackle the issue of sending volumes generated on local servers connected to X-ray detectors to remote dedicated servers for visualization. The local servers are freed to receive data at high frame rates and to perform local processing on the data using custom multi-GPU code. Such processing may involve, for instance, frame correction operations and even high performance tomography reconstruction [8]. In parallel, the remote servers receive the resulting volumes and are responsible for rendering them at full resolution, with interactive responsiveness.

* This is a joint work between the following groups of the Brazilian Synchrotron Light Laboratory: Sirius Scientific Computing (SSC – TVS, AZP, MLB, and EXM), Throughput Enhanced Processing Unit (TEPUI – FSF), and Beamline Software (SOL – DADA); in collaboration with the NVIDIA IndeX team (MN and AK). Corresponding author: Eduardo Xavier Miqueles (eduardo.miqueles@lnls.br).

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL05

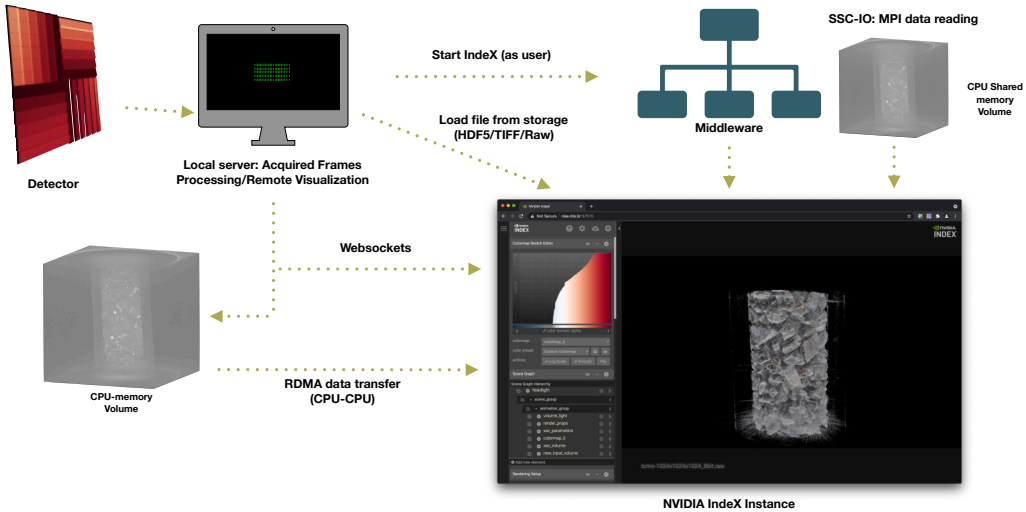

Figure 1: The proposed remote visualization workflow using the RemoteVis library to send volumes into NVIDIA IndeX for visualization, via RDMA data transfer.

The volume visualization solution we propose through RemoteVis provides much more than simple data transfer between nodes. We extended the library to serve as a platform for managing multiple instances of NVIDIA IndeX simultaneously, running in an HPC environment via job scheduling. Users of our APIs can transparently manipulate those instances with no more than a couple of C/C++ or Python functions calls. Furthermore, RemoteVis guarantees bidirectional data communication between NVIDIA IndeX and other softwares. This allowed us not only to send volumes, but also to retrieve portions of the volume that are being displayed for further processing. Moreover, NVIDIA IndeX provides two important functionalities that can be leveraged. First, the software implements the NVIDIA IndeX Accelerated Compute (XAC) facilities, through which custom functions can be created interactively for providing specific scientific visualization insights. Second, we are able to send pre-trained Deep Learning networks using RemoteVis to IndeX for inference. The software is then able to apply those networks instantaneously to the volume, thereby enhancing visualization via segmentation.

It is worth noting that Paraview [9] is an open source software that can use NVIDIA IndeX as a highly efficient rendering backend. We refrained from exploiting such integration to fully leverage the raw capabilities of IndeX in our workflows.

The paper is organized as follows. We first present the details about the RemoteVis library, including the capabilities of NVIDIA IndeX. We then briefly present some results before stating our concluding remarks.

THE REMOTEVIS LIBRARY

We may divide the RemoteVis library roughly into two major components. The first is an API written in C/C++ and Python whose primary goal is to perform data transfer/communication between NVIDIA IndeX and other softwares, which may be located in a same server or separated across an Ethernet network. The second is a middleware responsible for processing some of those API calls and for managing instances of IndeX, which may be running via a job scheduling mechanism such as slurm [10].

Figure 1 presents our visualization workflow combining RemoteVis and NVIDIA IndeX. Given a local server being operated by a beamline user for their experiment, the RemoteVis API is called to request a new IndeX visualization server instance. The RemoteVis middleware receives this request and initializes an instance on the beamline user's behalf in a remote server. All communication at this point is performed via TCP/IP socket transparently to the user of our API. While the NVIDIA IndeX instance is being initialized, the local server may be acquiring frames from the attached detector and/or conducting some processing on the received data, in order to produce the volume containing the sample of interest in CPU memory.

Once the sample volume is ready, yet another function call of RemoteVis performs two operations. First, the API asks the IndeX instance to prepare for receiving a new volume for data injection/rendering, using the software's websocket communication standard. Second, the volume is sent from the CPU memory of the local server directly to the CPU memory of the remote server running the IndeX instance via RDMA. When IndeX receives the volume, the software immediately injects the data into its rendering scheme by essentially transferring everything to multi-GPU memory.

RemoteVis is very flexible and its usage is not limited to transferring volumes from servers connected to detectors. Our API render volume function can be called with any 3D volume for remote rendering in CPU memory, regardless of the data origin. RemoteVis also provides another function to request the remote IndeX instance to directly load a file from

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL05

central storage. To further facilitate the use of RemoteVis by non-expert users, the library also incorporates a web interface through which the users can login and immediately start using an IndeX instance, without ever having to type a single line of code. In the next sections, we provide more details about NVIDIA IndeX and the components of RemoteVis.

NVIDIA IndeX

NVIDIA IndeX is a 3D volumetric interactive visualization platform that allows scientists and researchers to visualize and interact with massive data sets, make real-time modifications, and navigate to the most pertinent parts of the data, all in real-time, to gather better insights faster [1]. To achieve high efficiency and handle arbitrarily large data sets, IndeX divides a volume into smaller cubes and sends them for rendering across multiple GPUs and/or multiple nodes. IndeX uses a truly distributed database/compute environment to process the workload.

We implemented a plugin for data injection using IndeX's compute API, such that volumes are directly copied from CPU memory into the rendering pipeline for immediate visualization. Our plugin activates data injection after the RemoteVis library transfers the volume into the CPU memory of the server in which the IndeX instance is running, as will be detailed later.

The NVIDIA IndeX Accelerated Computing (XAC) interface enables scientists to rewrite the core data visualization routines at runtime. XAC-based programs are written in CUDA, compiled, and injected into the inner loops of the volume or surface raycasters to provide scientists with real-time visual feedback. We created XAC kernels for specific use cases of the Brazilian Synchrotron Light Laboratory and present some renditions in the results section (Figs. 2 and 3).

Visualization Server Middleware

NVIDIA IndeX ships with a web application built on top of the IndeX rendering library, consisting of a websockets server that receives JSON remote procedure calls from a React.js web viewer. The viewer displays the remotely rendered volume as an HTML5 video stream, with which the user interacts on their own web browser using mouse events. Those events are sent to the web server and effectively change the camera's position. We refer to this web application as a *visualization server instance* of NVIDIA IndeX in our work.

The RemoteVis library provides function calls to initialize, query, and shutdown instances of IndeX. The RemoteVis middleware is written in Python and processes those function calls, assigning a visualization server instance to the requesting user upon initialization. RemoteVis utilizes TCP/IP socket messaging between the user of our API and the middleware to ensure transparent client-server communication in a local Ethernet network. The visualization server instances may run directly on the server where the middleware resides as separate user processes or as scheduled slurm jobs, depending on the operation mode selected for the middleware. We detail later how slurm job integration works, but it is important to stress that regardless of the operation mode, the IndeX instance is started by the middleware as a process with system permissions of the requesting user. We thus ensure that all data access permissions are granted accordingly and respect the underlying HPC environment policies. Moreover, we assume the user has been previously authenticated into the local HPC environment and has been given the necessary privileges to use the RemoteVis API in their code.

The middleware starts the NVIDIA IndeX web server instance on a randomly chosen TCP/IP port that is then associated with the user. A URL is sent back through the API as the result of the initialization function for the user to access the IndeX instance on their web browser. The user may also invoke a query operation to retrieve the URL information later, or request the instance to be shut down programmatically.

Slurm Scheduling Integration

Since we propose RemoteVis as a framework for creating visualization workflows based on NVIDIA IndeX in an HPC environment, a common way of sharing resources in those settings is to make use of job schedulers such as slurm. When using the middleware in this operation mode, it starts visualization jobs via slurm and submits them to the job queues specified to run visualization server instances. Once again, those jobs receive the access permissions and slurm priority of the requesting user.

From the RemoteVis API perspective, the primary change in the slurm job operation mode is that the IndeX visualization server instance may not start immediately, depending on the state of the job queues. Hence, when the user requests an instance the middleware's default behavior is to create the slurm job and respond at once. Normally, the API function call is blocking, meaning that the user could readily send data for visualization once it returns, because the middleware waits for the instance to begin executing. This behavior changes when using the slurm backend, and the middleware returns the expected start date and time for the job instead of the URL. Currently, RemoteVis transfers the responsibility of verifying if the visualization job began to run to the user, who should use the API query function call to determine if the job has started.

When the IndeX instance job starts running, it communicates its current status to the middleware via RemoteVis. If the user queries the middleware at this point, the URL of the instance is returned as expected and data may be transferred to IndeX for visualization. It is worth noting that the NVIDIA IndeX job may be spawned on a server different than the machine in which the middleware resides, depending on the slurm configuration.

Since the user may not access the instance when the job begins its execution, possibly because the job started earlier than expected or the user forgot to check, the middleware automatically sends a shutdown command within a configurable grace period (usually 60 minutes) to release the computing resources for other jobs. The middleware also sends an estimated number of GPUs to slurm that are nec-

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL05

essary for rendering the volume, based on the size and data types requested by the user in the initialization function call. Those parameters are enforced during data transfer calls.

SSC-RDMA: RDMA-based Data Transfer Library

RemoteVis implements data transfer to and from NVIDIA IndeX using RDMA. RDMA is a low latency protocol for reading and writing information stored on CPU memory remotely, bypassing the CPU to achieve high transfer rates. It was originally proposed for Infiniband networks, but has since been introduced to Ethernet through RDMA over Converged Ethernet (RoCE).

We created a standalone C library called SSC-RDMA (Sirius Scientific Computing RDMA) to achieve high data transfer rates using RDMA, based on [11]. SSC-RDMA's primary goal is to transfer image volumes. Hence, the library provides two high level functions that can be called in the receiver and sender sides, respectively. Both functions accept two `char` arrays that are used to transfer the volume of interest with arbitrary size and a header containing information to describe the volume. We represent a 3D volume in C array format (column-major) and therefore cast any type of volume to a `char` array for generic data transfer. The size of the linearized in bytes array must be shared between sender and receiver out-of-band, to ensure that both sides allocate the data with the proper size before the transfer takes place. The `char` header array assumes a fixed size of 128 KB and carries application-dependent encoded information about the volume (e.g., dimensions, data type).

When sending a volume to the visualization server instance for rendering, the C/C++-level RemoteVis `render_volume` function accepts as input a `char` array containing the volume data, the volume information (size and data type), and the information about the instance obtained during initialization or query. The RemoteVis `render_volume` function then sends a websocket message to IndeX to have it prepare for receiving a new volume with the corresponding size in bytes. Simultaneously, the SSC-RDMA data transfer `ssc_send` function is invoked by the RemoteVis `render_volume` call and waits for our IndeX plugin to connect for data transfer using the `receive_volume` function, from the RemoteVis API. The latter uses in turn the generic `ssc_receive` function of SSC-RDMA and data is finally transferred from the local server to the remote server where IndeX is running. Afterwards, our plugin immediately asks the IndeX library to render the volume by injecting it from the CPU to the GPU memory.

To retrieve a volume from NVIDIA IndeX, the process is similar to the above. The exception is that the `retrieve_volume` function from RemoteVis requests IndeX to send the volume that is being rendered via SSC-RDMA.

Python API and Jupyter

RemoteVis provides full access to the underlying C/C++ functions from Python, which in fact are extremely similar. The primary difference between the functions is that the Python version accepts Numpy arrays holding the volumes for remote rendering. Our API can be be called directly from Jupyter notebooks and include a special function of RemoteVis that returns a screenshot of the current volume visualization using IndeX an RGBA numpy array.

Web Client for User-friendly IndeX Access

Even though the RemoteVis API is very simple to use, many users are not interested in coding, or don't know how to code, an application. We provide a user-friendly web client through which the user logs into the system using their credential of the HPC system, and then the RemoteVis API is called to start an instance on their behalf.

Our web client communicates with a web server written in Python/Django, which validates the user credentials on the HPC system and calls the RemoteVis IndeX initialization function upon success. Since the RemoteVis API is used throughout the process, all IndeX access is unified through the RemoteVis middleware, thereby providing great flexibility for designing visualization workflows.

We have further expanded the original React.js IndeX web interface to allow users to load volumes from disk. The user may navigate on a centralized repository file system using a custom React.js component, select a volume for loading and pass the corresponding string to our IndeX plugin. That volume is read from disk and sent to IndeX for rendering.

SSC-IO: MPI-based Efficient HDF5 File Reading and Writing

Since many synchrotron light sources have been adopting the HDF5 file format for storing various imaging formats, we implemented a library to efficiently read and write volumes into memory. This library is called SSC-IO and it was motivated by the fact that the supported way of reading and writing HDF5 files in parallel requires using the Message Passing Interface (MPI) standard. MPI has certain limitations that hinder its use with IndeX. In particular, IndeX implements an efficient pipeline for reading volumes from disk using parallel callbacks that are issued only for the visible portions of the volume. Those callbacks are issued by separate threads and work well for raw volumes. However, the HDF5 file format does not provide proper parallelization at the thread level, thereby decreasing performance

We designed SSC-IO to read volumes from HDF5 using MPI and storing the result in shared CPU memory. This is necessary because MPI spawns processes instead of threads to achieve parallelization. Once the volume is read, we implemented a custom loading callback plugin for IndeX that reads data directly from CPU memory. SSC-IO parallelizes writing a volume from CPU memory to disk in the same way. We designed the library in Python such that we can easily extend it to multiple HDF5 file structures.

RESULTS

In the next sections, we present some results obtained using RemoteVis and IndeX. In particular, we focus on briefly

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL05

evaluating RDMA data transfer and detailing some of the rendering features of IndeX.

RDMA-based Data Transfer

Table 1 presents some data transfer results using SSC-RDMA. We conducted 12 experiments using three servers, one IBM Power 9 and two NVIDIA DGX-A100s. The Power 9 is connected to the DGX-A100 through a high speed Ethernet switch at 50 Gb/s. The connection between the DGXs is via 100 Gb/s Ethernet. The results in the table are the average of 5 executions using float 32-bit volumes with 4 different sizes. As can be seen, SSC-RDMA is very efficient and requires roughly 20s to transfer over 100 GB of data (for volume with size 3072^3 voxels). The selected sizes reflect some typical volume sizes produced by fast detectors in a matter of seconds. As can be noted, SSC-RDMA achieves roughly 45 Gb/s data transfer rates, indicating that there is still room for improvement in our code, even though we are able to move large volumes without using centralized storage.

Table 1: RDMA-based data transfer times using SSC-RDMA. Three settings were tested between three different servers, one IBM Power 9 and two NVIDIA DGX-A100. The IBM Power 9 ($P9_{50}$) is connected at 50 Gb/s to the NVIDIA DGX-A100 while the connection between the DGX servers is at 100 Gb/s. The selected volumes are of type float 32 bits (4 bytes per voxel) and vary in number of voxels. All times are in seconds. For DGX to DGX, the destination server was the same as $P9_{50}$ to DGX.

# vx	$P9_{50}$ to DGX	DGX to $P9_{50}$	DGX to DGX
1024^3	0.87 ± 0.01	1.13 ± 0.20	0.86 ± 1.48
1536^3	2.58 ± 0.09	3.00 ± 0.65	2.53 ± 0.47
2048^3	5.99 ± 0.09	7.17 ± 1.47	5.98 ± 1.12
3072^3	20.30 ± 0.26	21.89 ± 4.90	19.16 ± 3.66

XAC Rendering Kernels

We have implemented three different rendering techniques using XAC based on some use cases of LNLS. The first XAC kernel (S1) performs simple shading using local information such as gradient calculation on the volume. The second kernel (S2) simulates single scatter lighting and requires some extra memory for rendering, but produces shadow effects that improve visualization. The third is the most computationally demanding and is based on the ambient occlusion algorithm [12] (S3), requiring pre-computation to generate an auxiliary volume to determine shadow values. Note that all computation was done directly inside IndeX using the XAC technology.

Figures 2 and 3 presents two samples of X-ray computed tomography imaged at LNLS and redered using XAC. The first figure compares basic rendering with (S1) and (S3), while the second compares (S1) and (S2). The original volume of the silica bead experiment (Fig. 3) can be seen segmented in Fig. 4.

NVIDIA TensorRT CNN Inference

As previously stated, NVIDIA IndeX provides integration with TensorFlow/NVIDIA TensorRT such that pre-trained deep neural networks may be used to perform on-demand inference on the visible portion of the volume. We have devised softwares at LNLS capable of producing NVIDIA TensorRT-optimized inference engines from CNNs created for segmenting volumes, which can then be used with IndeX to provide advanced data interpretation.

Figure 4 presents an example of those results. In that case, we trained a 2D U-net [13] model to segment the phases of a silica bead fluid flow experiment containing four phases of interest (beads, water, gas, and background) using our custom softwares and then sent the resulting TensorRT engine for IndeX using RemoteVis. In that case, RemoteVis provides yet another function capable of taking a serialized engine and sending it to IndeX via SSC-RDMA for inference.

Point Cloud Rendering

NVIDIA IndeX was designed not only to render regular volumes, but also irregular data such as meshes and particle volumes (i.e., point clouds). We have incorporated into RemoteVis the capability of rendering point clouds submitted by the user similarly to how it is done for regular volumes. The corresponding function call accepts an array holding the `XYZ` coordinates of the points and another with the values of each one. Then, RemoteVis invokes RDMA data transfer for each of those arrays into the server where the IndeX instance is running and requests the software to render the point cloud as a particle volume. The points are rendered as spheres with radii proportional to the point values. The API is accessible from C/C++ and Python as always. Figure 5 depicts an example of point clouds being rendered from data obtained for 3D reciprocal space mapping, a common experiment in synchrotron light sources.

CONCLUSIONS

We presented the RemoteVis library as a framework for creating 3D visualization workflows using NVIDIA IndeX as the volume rendering backend. RemoteVis provides a data transfer mechanism via RDMA to allow volumes to be transferred from local servers to remote servers dedicated to visualization. The data in local servers may be acquired from X-ray detectors or simply produced by some other procedure. RemoteVis provides auxiliary functions that can load HDF5 files directly into IndeX very efficiently using MPI. Also, our library allows portions of the volume being rendered to be retrieved from IndeX into other softwares for further processing.

We also exploit in RemoteVis very useful techniques implemented in IndeX. First, we are able to create insightful rendering functions using the IndeX Accelerated Compute technique to provide advanced visualization for certain scientific cases. Second, we are able not only to render regular volumes but also point clouds int IndeX using RemoteVis, also via RDMA data transfer. Finally, we incorporate NVIDIA

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL05

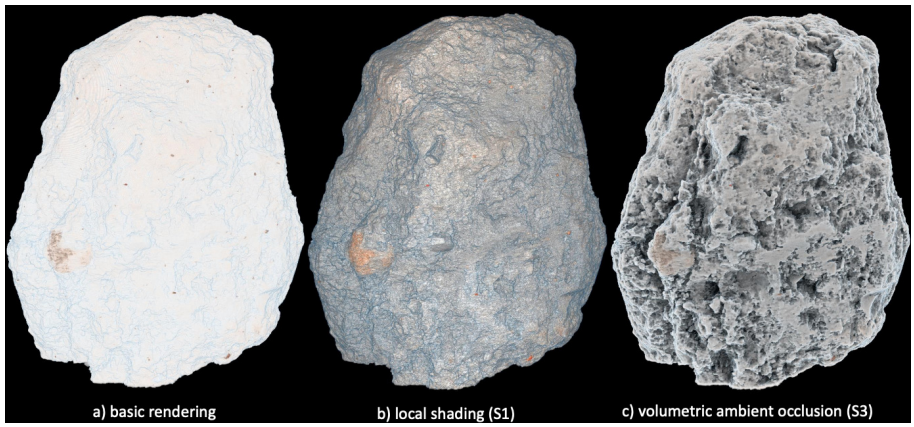

Figure 2: Soil sample renditions using IndeX XAC operators. (a) The original volume with basic rendering and no shading. (b) The volume rendered with XAC local shading (S1). (c) The more advanced XAC shading scheme with ambient occlusion (S3). Data courtesy: MOGNO beamline/Sirius, LNLS/CNPEM.

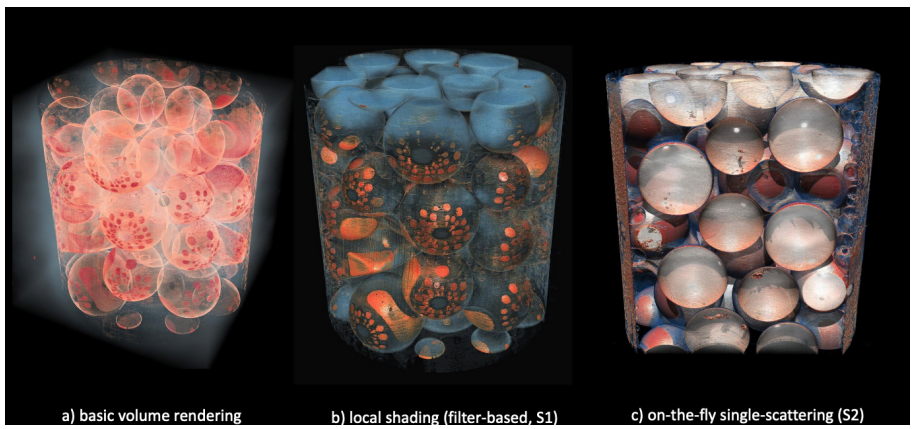

Figure 3: Silica bead fluid flow experiment renditions using IndeX XAC operators. The original image with segmentation of the gas phase is depicted in Fig. 4 (left). Data courtesy: MOGNO beamline/Sirius, LNLS/CNPEM.

TensorRT-optimized pre-trained CNNs into IndeX, such that our models provide on-the-fly segmentation of the data that is used to enhanced visualization.

RemoteVis is very well suited for use in synchrotron light sources, although it is not limited to those use cases. RemoteVis and IndeX can be used in HPC environments where large data sets are produced and require fast and easy to use tools for visualization.

ACKNOWLEDGEMENTS

We would like to acknowledge the Brazilian Ministry of Science, Technology, and Innovation (MCTI) for funding this work, through the Brazilian Center for Research in Energy and Materials (CNPEM). We would also like to thank the MOGNO and EMA beamline groups from Sirius/LNLS-CNPEM for the data used in this paper.

REFERENCES

[1] *NVIDIA IndeX*, Sep. 2021. https://developer.nvidia.com/nvidia-index

[2] *Neuroglancer*, Sep. 2021. https://github.com/google/neuroglancer

[3] N. Sofroniew *et al.*, *Napari/napari: 0.4.11rc3*, version v0.4.11rc3, Sep. 2021. DOI: 10.5281/zenodo.5349926. https://doi.org/10.5281/zenodo.5349926

[4] B. Schmid, J. Schindelin, A. Cardona, M. Longair, and M. Heisenberg, "A high-level 3d visualization api for java and imagej," *BMC Bioinformatics*, vol. 11, no. 1, p. 274, 2010. DOI: 10.1186/1471-2105-11-274. https://doi.org/10.1186/1471-2105-11-274

[5] A. Fedorov *et al.*, "3d slicer as an image computing platform for the quantitative imaging network.," eng, *Magn. Reson. Imaging*, vol. 30, no. 9, pp. 1323–1341, Nov. 2012, ISSN: 1873-5894 (Electronic); 0730-725X (Print); 0730-725X (Linking). DOI: 10.1016/j.mri.2012.05.001.

[6] *Dragonfly*, Sep. 2021. https://www.theobjects.com/index.html

[7] *OpenInventor*, Sep. 2021. https://www.openinventor.com

[8] E. X. Miqueles, G. Martinez Jr., and P. P. Guerrero, "Fast image reconstruction at a synchrotron laboratory," in *SIAM Conf. Parallel Proc. Scientific Comp.* 2020, pp. 24–34. DOI: 10.1137/1.9781611976137.3. https://epubs.siam.org/doi/pdf/10.1137/1.9781611976137.3. https://epubs.siam.org/doi/abs/10.1137/1.9781611976137.3

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL05

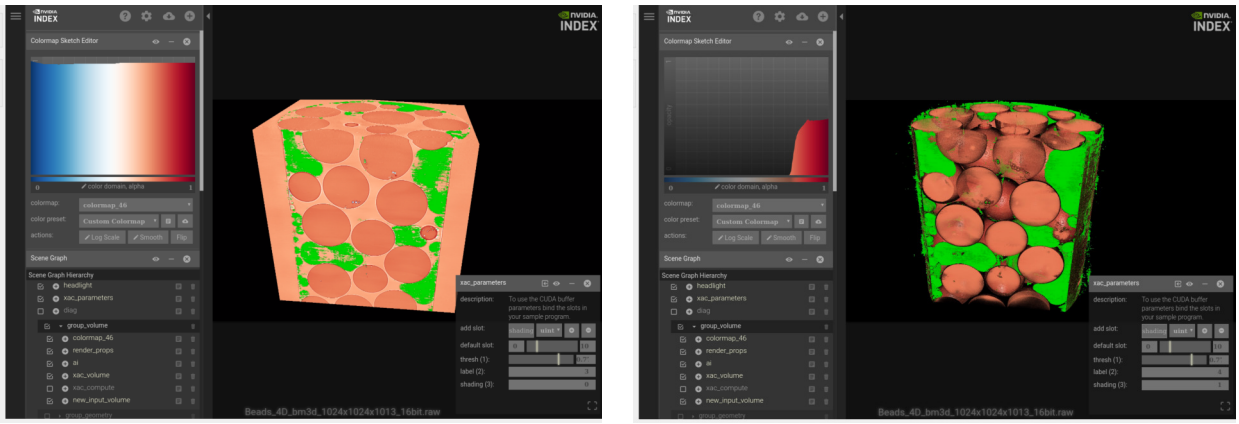

Figure 4: Integration between a pre-trained U-net segmentation model, optimized with NVIDIA TensorRT, and IndeX for visualization. The TensorRT inference engine is called to segment the gas phase of the sample (left) and the result is immediately used in the 3D rendering (right). Data courtesy: MOGNO beamline/Sirius, LNLS/CNPEM.

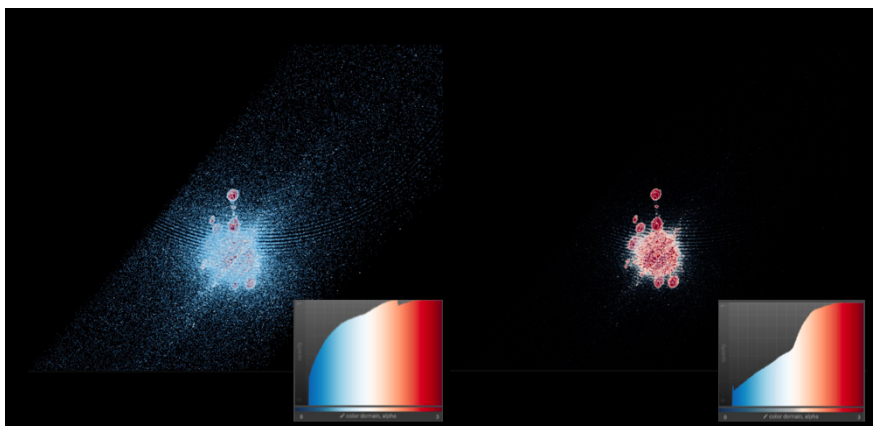

Figure 5: Point cloud renditions by NVIDIA IndeX as a particle volume. The `XYZ` and point value data are transferred by RemoteVis using SSC-RDMA and the points are rendered as spheres with radii proportional to their values. The user can then select which points to view based on their radii (left) by simply altering the considered colormap (right). Data courtesy: EMA beamline/Sirius, LNLS/CNPEM.

[9] U. Ayachit, *The ParaView Guide: A Parallel Visualization Application*. Clifton Park, NY, USA: Kitware, Inc., 2015, ISBN: 1930934300.

[10] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60, ISBN: 978-3-540-39727-4.

[11] T. Bedeir, *Basic flow control for rdma transfers*, Accessed on Sep 4, 2021, Jan. 2013. `http://www.hpcadvisorycouncil.com/pdf/vendor_content/basic-flow-control-for-rdma-transfers.pdf`

[12] G. Miller, "Efficient algorithms for local and global accessibility shading," in *SIGGRAPH*, ser. SIGGRAPH '94, New York, NY, USA: ACM, 1994, pp. 319–326, ISBN: 0897916670. DOI: `10.1145/192161.192244`. `https://doi.org/10.1145/192161.192244`

[13] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., Cham: Springer International Publishing, 2015, pp. 234–241, ISBN: 978-3-319-24574-4.