Faster X-Ray Computed Tomography in Real-world Dynamic Applications

Proefschrift

ter verkrijging van
de graad van doctor aan de Universiteit Leiden,
op gezag van rector magnificus prof.dr. S. de Rijcke,
volgens besluit van het college voor promoties
te verdedigen op woensdag 4 februari 2026
klokke 14.30 uur

door
Adrianus Benedictus Maria Graas
geboren te Hoorn
in 1989

Promotor:

Prof.dr. K.J. Batenburg

Co-promotor:

Dr. F. Lucka (Centrum Wiskunde & Informatica)

Promotiecommissie:

Prof.dr. M.M. Bonsangue

Prof.dr.ir. F.J. Verbeek

Dr. L. Cao

Prof.dr. V. Kolehmainen (University of Eastern Finland)

Dr. N. Viganò (Commissariat à l'énergie atomique, CEA-IRIG)

The research presented in this dissertation was carried out at Centrum Wiskunde & Informatica (CWI) in Amsterdam. Financial support was provided by the Netherlands Organisation for Scientific Research (NWO), project numbers 613.009.106, 613.009.116, and VI.Vidi.223.059.

The illustrations on the cover and invitation depict the time-evolution of one bubble through a gas-solids fluidized bed, as measured by three X-ray detectors. See chapters 2 and 5.

Contents

Chapter 1

Introduction

In November 1895, Wilhelm Conrad Röntgen acquired the world's first X-ray photograph, now known as a *radiograph*, on a piece of paper painted with barium platinum cyanide [1]. The exposure to the new type of light, leading to the famous picture of the bones in the hand of Anna Bertha Ludwig, took *about 15 minutes*. Today, more than 125 years later, modern X-ray imaging technology can capture such radiographs with framerates up to hundreds or thousands Hertz, almost a million times faster in comparison to Röntgen's original measurement.

Computed Tomography (CT) may very well be the most influential invention originating from Röntgen's discovery. Also known as "computerized (axial) *tomography*" (*tomos*: slice, *graphein*: to write), CT enables non-intrusive diagnostics of human subjects, as well as inspection and observation in the experimental sciences. The principle of CT is to compute a digital representation of a 3D interior (a human or object) via a set of radiographs. This set of radiographs is obtained via scanning the object from different angles. In 1973, Hounsfield's iterative algorithm, delivered on a Minicomputer with 32 kilobyte of memory in the first commercial CT scanner called the EMI, took *about 5 minutes* to transform 28,800 data points into a 80×80 polaroid of a cross-sectional slice of the patient's brain [2]. Nowadays, more than 50 years later, computing a CT image from 1 billion data points typically takes anywhere from *a second to several minutes* on a personal computer, depending on the algorithm used and the size of the data.

After all these technological advances, one may get the impression that CT reached its full potential or that it does not need to be much faster. Many medical and research applications using CT permit a few minutes of waiting time while algorithms are executed, for example, between the CT scan of a patient and the doctor's diagnosis—or, similarly, between the scan of an object in a laboratory and a researcher's evaluation. In this thesis, however, we will see that CT is not nearly as fast as we would like it to be. One aspect is the unrealized potential that *interaction with the scan* holds. When CT reconstructions would

be repeated during a scan, for example, a scan could be terminated precisely when enough information is collected—reducing laboratory occupation or limiting the radiation exposure to the subject. Another motivation is given by the recent surge of data-driven algorithms, for example in dynamic tomography, which require many consecutive CT reconstructions to be computed efficiently. Even though CT by itself is fast, *faster* CT could enable a new category of real-time and dynamic tomographic applications.

This dissertation uses the properties of sequential and dynamic data in X-ray applications to increase the speed of CT algorithms. It discusses a set-up for fast data acquisition, numerical implementations for faster algorithms, but also explores how, with artificial intelligence, data sets can improve fast X-ray Computed Tomography algorithms in practical, real-world applications. This chapter first introduces the preliminaries and background, and then formulates the research questions associated with the subsequent chapter-by-chapter publications.

1.1 Primer on Computed Tomography

Computed Tomography (CT) is a non-invasive technique to form a three-dimensional (3D) image of a subject or object using X-ray radiation. To suit different applications, there exist many variations of the technique and it is broadly researched within the fields of mathematics, computer science, physics, and applied sciences. In all its variations, the essence of the technique is to solve what is known as an *inverse problem*. To get an impression of what this means, assume an opaque *Object* that we would like to obtain a 3D image of, as well as access to an *X-ray Set-up* that can be used to obtain *Radiographs* of the object:

$$\text{Object} \xrightarrow{\text{X-ray Set-up}} \text{Radiographs}. \tag{1.1}$$

In CT, the goal is to reverse this relation, i.e., to find a digitalized 3D representation of the unknown object interior via its radiographs:

$$\text{Radiographs} \xrightarrow{\text{CT Algorithm}} \text{Object}. \tag{1.2}$$

To recover the *Object*, the *CT Algorithm* needs to model the geometry of the X-ray set-up, and take into account the X-ray physics that lead to the radiographs. The inverse problem of X-ray CT is known to be ill-posed, meaning that a solution may not exist, may not be unique, and may be sensitive to noisy data. Computed Tomography is a prototypical example of an inverse problem in the area of medical imaging, due to its long history, extensive use, and the comparatively simple X-ray physics.

Absorption contrast tomography X-rays are a frequency range of the electromagnetic spectrum (i.e., a type of light) with a wavelength shorter than ultraviolet light. X-rays that pass through a medium can interact with its atoms in several ways, such as absorption (the photo-electric effect), scattering, or phase shifts. During absorption and inelastic scattering, the energy of X-rays is transferred to the atoms in the material, and this reduces the intensity

of the X-ray beam. This principle is the basis from which most Computed Tomography algorithms are derived. Interactions that are not modeled can lead to noise or artifacts in the algorithm's outcome. The reader is referred to the textbook by Buzug [3] for an in-depth treatment of the X-ray physics.

A CT algorithm works via the use of multiple *absorption images* or *projections*, which are in fact just radiographs processed in a way that the image intensity relates linearly to absorption inside the object (this is discussed as the measurement principle hereafter). Each absorption image must be taken from a different angle of the object. Absorption images are generally monochromatic (i.e., in grayscale color), since energy-integrating X-ray detectors are unable to distinguish between the individual energies of photons.

The digital representation that is the outcome of a CT algorithm, called a *reconstruction volume*, is a 3D image describing the local X-ray absorption inside the object—indeed, when the materials inside it have similar X-ray absorption characteristics, this results in a low contrast 3D volume. With sufficient time and without limitations on radiation exposure, hundreds to thousands of angular measurements may be required to resolve small-scale details inside the object.

The measurement principle As mentioned, tomographic algorithms do not work with raw detector radiographs, but instead require an initial preprocessing of the radiographs given by the detector. The radiographs, in an idealized case, are described by Beer-Lambert's law: For a monochromatic pencil beam [3], i.e., a narrow and collimated X-ray source, it describes the relation between the radiation intensity I^0 at the source, and its decay through an object with attenuation μ along a ray l_i as

$$I_i = I_i^0 \exp\left(-\int_{l_i} \mu(\eta)\,\mathrm{d}\eta\right), \tag{1.3}$$

where η denotes the position along the ray, $\mu(\eta)$ the attenuation coefficient at η, and I_i the remaining intensity measured at the detector pixel.

X-ray intensities are described with a discrete number of *detector counts*. These values represent an equivalent of the photons measured in the pixel [4]. Assuming that the attenuation of air is negligible, measurements taken in air under the same experimental conditions form a *flatfield*, and can be used to obtain I^0. With I^0, the *absorption image* or *projection* is defined as

$$\log\left(\frac{I^0}{I}\right) = \int_{l_i} \mu(\eta)\,\mathrm{d}\eta. \tag{1.4}$$

Note that the projection is linearly related to $\mu(\eta)$, which describes all attenuating objects along a ray l_i.

Cone-beam CT As an example, one of the most common configurations of CT is presented, namely with a cone-beam source, i.e., a three-dimensional divergent X-ray beam, and a flat-panel X-ray detector. The source and detector describe a full circular trajectory

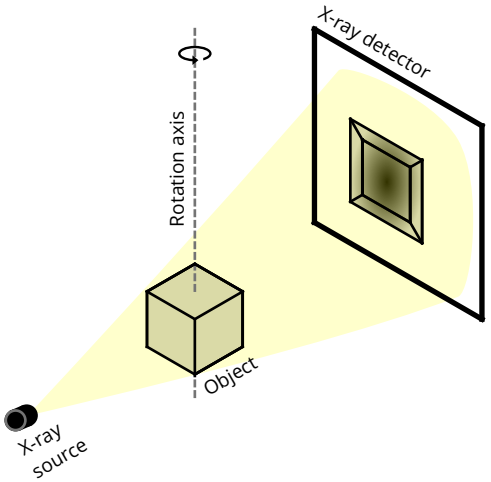


Figure 1.1: A CBCT set-up: Radiographic projections of the opaque interior of *Object* are acquired via illumination of the object with a divergent X-ray beam. The set-up or object is rotated around a central axis to obtain the radiographs from equidistant angles.

around the object to acquire an equiangular range of absorption images. This set-up geometry, which is the "forward problem" of equation 1.1, is termed circular *cone-beam CT* (CBCT) and visualized in figure 1.1.

The reader is referred to the textbooks of Hansen [4] and Kak & Slaney [5] for the mathematics and algorithms of tomography, in the following only a brief outline of the mathematics is presented. Mathematically, the inverse problem (equation 1.2) amounts to solving the linear equation

$$\mathbf{A}x = y. \tag{1.5}$$

for the unknown volume x. Here, y is a stacked vector, assembling the pixels from all absorption images and $\mathbf{A}$ denotes the linear X-ray operator. Each i-th row of $\mathbf{A}$ corresponds to a discretized line integral, running from the X-ray source to detector pixel associated with the index i. Most values in $\mathbf{A}$ are empty, as each line integral uses only a few voxels in x to compute the pixel i. The line integral and numerical implementation will be discussed later in the introduction (section 1.6).

Feldmann-Davis-Kress algorithm To solve equation 1.5, with $\mathbf{A}$ built using a cone-beam geometry, there exist many different CT algorithms. A classical example is the Feldkamp-Davis-Kress (FDK) algorithm [6] which was developed for cone-beam geometries and enjoys popularity due to its fast execution [5, 7]. The algorithm can be formulated as:

$$x^{\star}_{\mathrm{FDK}} := \mathbf{A}^T(y \circledast f), \tag{1.6}$$

with $x^{\star}_{\text{FDK}}$ denoting the FDK solution. The algorithm consists of just two steps: The first is to convolve the absorption images y with a high-pass filter f to reduce the low-frequency bias in the measurements. The second step executes the mathematical adjoint $\mathbf{A}^T$ of the forward X-ray operator. Since $\mathbf{A}$ contains integrals along the paths of the X-rays, $\mathbf{A}^T$ "smears out" the filtered absorption images along the original X-ray paths in the 3D space of the object.

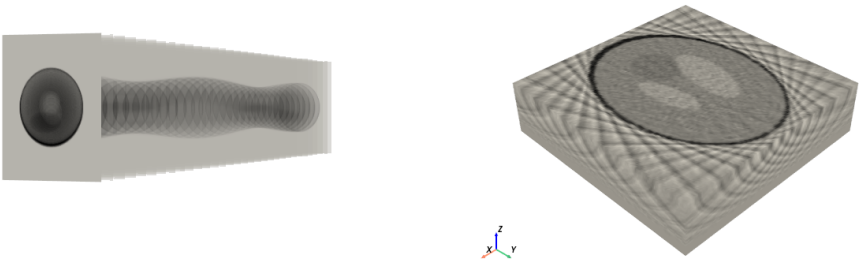


Figure 1.2: Reconstruction of the 3D *Shepp-Logan* phantom. On the left is a stack of 32 absorption images y, uniformly sampled from $[0, 2\pi)$ angles using the CBCT geometry of figure 1.1. The FDK reconstruction $x^{\star}_{\text{FDK}}$ with the Ram-Lak filter [5], on the right, is of low quality, due to the low number of angles.

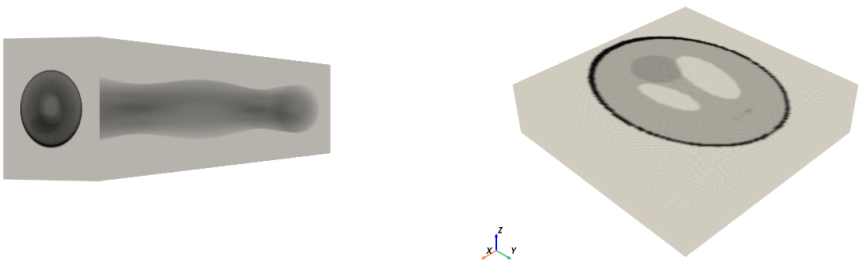


Figure 1.3: Reconstruction of the Shepp-Logan phantom of figure 1.2, now repeated using 256 absorption images. This removes the streaks and additionally recovers three small low-contrast ellipsoids in the bottom-right of the phantom.

Figure 1.2 gives an example of a numerically simulated CT reconstruction using a 3D phantom containing ellipsoid spheres of mixed intensities. The angular stack of simulated absorption images y is shown on the left. On the right, the 3D outcome of the FDK algorithm is displayed with a volume that is sliced through its center. This slice cuts three of the ellipsoids in the volume. However, because of the low number of absorption images, the FDK reconstruction also shows streaks throughout the volume. In figure 1.3, the simulation is repeated with 256 absorption images, demonstrating that the image quality is strongly de-

pendent on the number of absorption images. In section 1.5, we will continue the discussion on sparse-view geometries.

Simultaneous Iterative Reconstruction Technique (SIRT) For later reference, we also present SIRT, an example of an iterative, algebraic reconstruction technique. Whereas the FDK algorithm is an analytical technique, *algebraic* reconstruction techniques are a class of CT algorithms that discretize x on a volumetric grid and solve equation (1.5) algebraically. Hounsfield's first iterative CT algorithm is an example of such technique. Compared to a direct method, such as the FDK algorithm, algebraic methods are better suited to sparse-angle or noisy data [4]. A method that is commonly used nowadays, is SIRT [8]. SIRT reformulates equation (1.5) as a constrained weighted least-squares minimization problem, i.e.,

$$x^{\star}_{\mathrm{SIRT}} = \arg\min_{x \in \mathcal{C}} \ \|\mathbf{A}x - y\|^2_R \,, \tag{1.7}$$

where $x^{\star}$ is the least-squares solution, and $\mathcal{C}$ includes a set of constraints. An example would be the box constraints $0 \leq x \leq 1$, which restrict the reconstructed quantity to a percentage (Chapter 2). The norm $\|\cdot\|_R$ is weighted according to a diagonal matrix R, containing the reciprocals of the row sums of $\mathbf{A}$.

The optimization problem in equation (1.7) is solved iteratively using gradient descent, preconditioned with a diagonal matrix of column sums C. The algorithm proceeds with update steps

$$x^{(k+1)} = \Pi_{\mathcal{C}} \left(x^{(k)} + C\mathbf{A}^T R \left(\mathbf{A}x^{(k)} - y \right) \right), \tag{1.8}$$

where $x^{(k)}$ denotes the k-th iterate. $\Pi_{\mathcal{C}}$ denotes the orthogonal projection onto the constraint set, which effectively clips $x^{(k+1)}$ to $[0, 1]$, and sets $x^{(k+1)}$ to zero in the masked area. To prevent fitting noise, and because of a model mismatch in Eq. (1.5), SIRT is usually stopped at a fixed number of iterations.

1.2 Applications

Projectional radiography X-ray CT is found useful for a plethora of medical, industrial and scientific applications. Before we explore these, it is good to mention that for certain applications, radiographs alone may already provide sufficient information. For example, in Fig. 1.3, the contrast in the radiographs already reveal the edges of the ellipses, which, under additional assumptions, can be used to determine their volume. For X-ray fluoroscopy, non-destructive testing or defect inspection, similar contrast on radiographs can also provide sufficient information for the task at hand. One example is the set-up of Chapter 2, wich is used simultaneously for projectional radiography of bubbles, to extract e.g., their gas holdup, and for CT, to compute the bubble shapes. At the same time, radiographs can also be used to assist with CT: Medical CT scanners can acquire "scout views" of the body, to help localize a scanning region inside of the patient. Also in laboratory set-ups, the angular radiographs are often visualized during the scan, for example to reassure that the object remains in view, that the set-up settings (calibration, source voltage) are suitable, and to

track dynamics occurring in the object. Methods that work directly on radiographs, e.g., the denoising method that we introduce in Chapter 5, have the advantage that they can be extended to radiography applications.

Medical CT The traditional setting of CT is in hospitals, where it provides a fast and comparatively inexpensive imaging solution. Here the 3D reconstruction of a part of the human body can help with, e.g., the precise localization of a tumor or blood clot, or visualization of a bone fracture. To improve contrast, CT is sometimes used with iodinated contrast agents. However, it can also be used in conjunction with other modalities, such as MRI, which provides better contrast for soft tissues.

In medical CT scanners, a highly-streamlined two-step acquisition-reconstruction procedure obtains a single, three-dimensional computer representation of the human body. Medical CT scanners are equipped with a gantry, consisting of an opposing X-ray source and flat-panel detector, which enables capturing large cross-sectional parts of the body with a single orbit of the gantry. To guarantee safe and optimal use, manufacturers often enclose the software and hardware necessary for tomographic computations internally in the machine, and output the reconstruction as a slice-by-slice volume directly after the scan has proceeded.

Sequential CT In scientific and industrial applications, e.g., biology, life sciences, material sciences or engineering, a more fine-grained control over acquisition and reconstruction is needed. Experimental set-ups, such as the FleX-ray lab discussed in Chapter 3, allow laboratory technicians to tailor the scanning geometry on a per-experiment basis, and to retrieve the raw detector data after the acquisition. Rather than a single rotation, these devices can perform multiple continuous rotations, after which a *sequence of CT reconstructions* can be computed. To convey an impression, Figure 1.4 shows a few examples from application areas that use sequential reconstructions. Broadly speaking, sequential CT is common in the following areas:

- **Dynamic tomography**: A field of great interest in the scientific imaging is to recover the interior of a dynamic object or process, i.e., the tomographic data must be resolved in time. Prototypical examples are: the dynamics of multi-phase fluids (Chapter 2 and 3), dissolution of tablets, rising of dough, and combustion processes.

- **In-line tomography**: Here, a fixed measurement protocol is applied to a series of similar objects, for example, aligned on a conveyor belt. This scenario is more common in industrial applications, such as baggage screening, product inspection, or foreign object detection.

- **Explorative tomography**: Rather than dynamics that are inherent to the object, the scanning geometry is manipulated while the scan is ongoing. An example is to bring the source and object closer together, which increases the spatial resolution of the data and allows zooming into the sample. Another example is a *tiled scan*, stitching

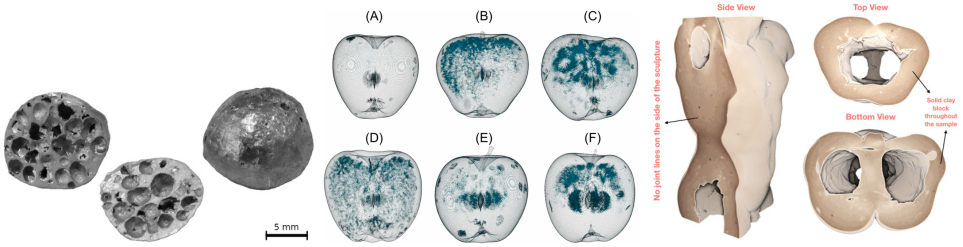


Figure 1.4: Three examples from application areas using sequences of tomographic reconstructions. **Left**: observation of pore morphology under compressive loading, figure reproduced from Vopalensky *et al.* [10], published under a CC BY 4.0 license. **Middle**: detecting internal browning in apple tissue, figure reproduced from Wood & Schut *et al.* [11] published under a CC BY 4.0 license. **Right**: searching for finger prints and tool marks in a terracotta mini sculpture, figure reproduced from Coban *et al.* [9], published under a CC BY 4.0 license.

multiple high-resolution reconstructions together to assemble a single reconstruction. Explorative tomography is useful for cultural heritage research and preservation [9].

1.3 Tomographic pipelines

To facilitate all these applications, scientists and laboratories often use certain workflows to process the radiographs and reconstructions. Unlike medical CT, this does not take place inside the scanning device, but on the researcher's personal computer or on computation facilities, with a number of consecutive computational or procedural steps that we will call a *tomographic pipeline* [12]. For our purposes, the pipeline is defined such that it includes the acquisition, pre-processing, reconstruction, and visualization/analysis of the sequential data that passes through it. A few practical examples of pipelines are:

- **Spatio-temporal CT reconstruction**: A timeframe-by-timeframe CT reconstruction, with, e.g., one full rotation of the scanner per timeframe. In Chapter 2, we will see that this can result in a high-resolution "3D movie", and allows studying and segmenting bubbles in a multi-phase flow.
- **Real-time reconstruction**: A region-of-interest of the volume (i.e., not in fully-3D, but in a part of the volume), can be produced in milliseconds, thus at a high framerate, during the experiment using the RECAST3D software package. A scientist can interact with the RECAST3D graphical user-interface to reorient and manipulate multiple slices to quickly visualize certain parts of the object (Chapter 3).
- **On-the-fly reconstruction for machine learning**: A machine learning algorithm needs to see a large quantity of example reconstructions before it can be applied to unseen data. When reconstruction data sets are too large for storage in computer memory, a pipeline can generate these reconstructions on-the-fly from the data set of radiographs (Chapter 4).

The images passing through tomographic pipelines generally possess a high degree of consistency. In a slow-moving fluid, for instance, the acquired data is temporally continuous, and in in-line inspection images are a deformed or modified version of a target template. This requires that acquisition and reconstruction settings (e.g., radiation intensity, detector configuration, reconstruction algorithm), are fixed during the experiment.

On-line and real-time tomography Most tomographic pipelines are *off-line*, which means that radiographs are stored after scanning and that computational steps for CT are ran at a later point in time. In off-line pipelines, time constraints are usually dictated by the set-up or by the physics (e.g., enough photons must pass through the object, or the scanning must be fast enough to follow the physics). In *on-line* tomographic pipelines, the goal is to perform the computational steps concurrently with the scan. On-line pipelines can also be *real-time*, meaning that they aim to minimize the lag between the experimental physics and the visualization component at the beginning and end of the pipeline (Chapter 3). In scientific applications of tomography, real-time usually indicates a delay of milliseconds to seconds.

The pinnacle of real-time tomography would be to use the information gained from reconstructions in the pipeline as (automated) feedback during the scanning process, which is referred to as *steering*. Steering is particularly important when the success of the experiment depends on environmental parameters, for example, when certain physical phenomena must be reproduced under unstable conditions, e.g., with a precise temperature or chemical balance. Steering then allows a laboratory scientist or algorithm to validate or tune the state of the experiment, e.g., by raising the temperature. With steering implemented, obtaining high-quality reconstructions avoids the typical cycle of trial-and-error experiments, which can be costly and time-consuming in X-ray facilities with scarce resources.

1.4 Optimizing tomographic pipelines

X-rays are known for their suitability in fast 3D imaging techniques. Since X-rays are high-energy electromagnetic waves, they propagate at almost the speed of light through any medium. Therefore, the imaging speed with X-rays is often limited by the rate at which detector instruments can operate. Certainly not every imaging task can be optimally addressed with X-rays, for example, due to the radiation damage that X-rays inflicts on biological samples, or due to limited photon flux (treated in the next section). However, the fast underlying physics makes CT uniquely suitable for real-time and high-speed dynamic imaging problems. In comparison, the scanning speed of MRI (Magnetic Resonance Imaging) is constrained by the relaxation time of protons (milliseconds to seconds), and the scanning speed of ultrasound imaging is ultimately limited by the speed of sound in biological tissue (microseconds to milliseconds).

In the previous section, we explained the concept of tomographic pipelines, and discussed their different applications and settings. A closer look at the use cases hints at how fast we would like tomographic pipelines to be, given ideal circumstances. Fast would be “fast

enough" when…

- …a set-up can follow the concurrent physics with a sufficiently high temporal resolution (for dynamic CT in an off-line pipeline);
- …a human or steering algorithm is able to respond timely in an ongoing experiment (for real-time CT);
- …a sufficient amount of data can be generated not to throttle a neural network training process (in an on-the-fly data sampling pipeline).

To accomplish these goals, it is not unusual to make large concessions inside the tomographic pipeline. A slow-moving fluid may require a pipeline operating at 10 Hz framerate, human observation perhaps 30 Hz, and a neural network training tasks (section 1.7) several hundreds to thousands of samples per second. To achieve these requirements, several or all components in the tomographic pipeline must act closely together. For example, a set-up may have to rotate faster than what is ideal, or a reconstruction may need to be performed at a lower resolution, or a machine learning algorithm must work with a less-than-ideal number of examples.

In the next sections we will discuss how different pipeline components contribute to faster X-ray CT. In section 1.5 and 1.6, fast scanning and fast reconstruction techniques are discussed. In 1.7 and 1.8 self-supervised deep learning is treated.

1.5 The trade-offs inside fast X-ray imaging set-ups

There are two important qualities of X-ray set-ups to be considered for fast tomographic imaging. The first is the detector framerate: A higher framerate enables a higher temporal resolution, but simultaneously leads to noisier radiographs. The second is the rate of angular acquisition, i.e., the time that it takes for the set-up to acquire all scan angles. Faster angular acquisition can increase the temporal resolution of a dynamic CT reconstruction, but can reduce the quality of each individual timeframe. Figure 1.5 illustrates three different types of scientific X-ray imaging set-ups. We will first study these set-ups, and then further discuss the noise levels and number of angles per rotation.

X-ray set-ups The first set-up is the FleX-ray laboratory at Centrum Wiskunde & Informatica in Amsterdam. The laboratory features a micro-CT scanner with a PerkinElmer Dexela 1512NDT detector. This detector connects to a single cable, and is capped to 26 frames per second via its Camera Link interface (opposed to a slower gigabit GigE Vision interface). Its framerate can be increased to 86 Hz by reducing the resolution of the data via binning of 4×4 pixel regions. Assuming a safe rotation speed of one cycle per second, the set-up can then reconstruct an object once every half-rotation using information from 172 angles.

The facility in the upper right of Figure 1.5 shows a beamline, i.e., an experiment station, in the Canadian Light Source synchrotron. Synchrotron light sources are at the forefront of developing tomographic pipelines. These facilities use a (cyclic) particle accelerator and

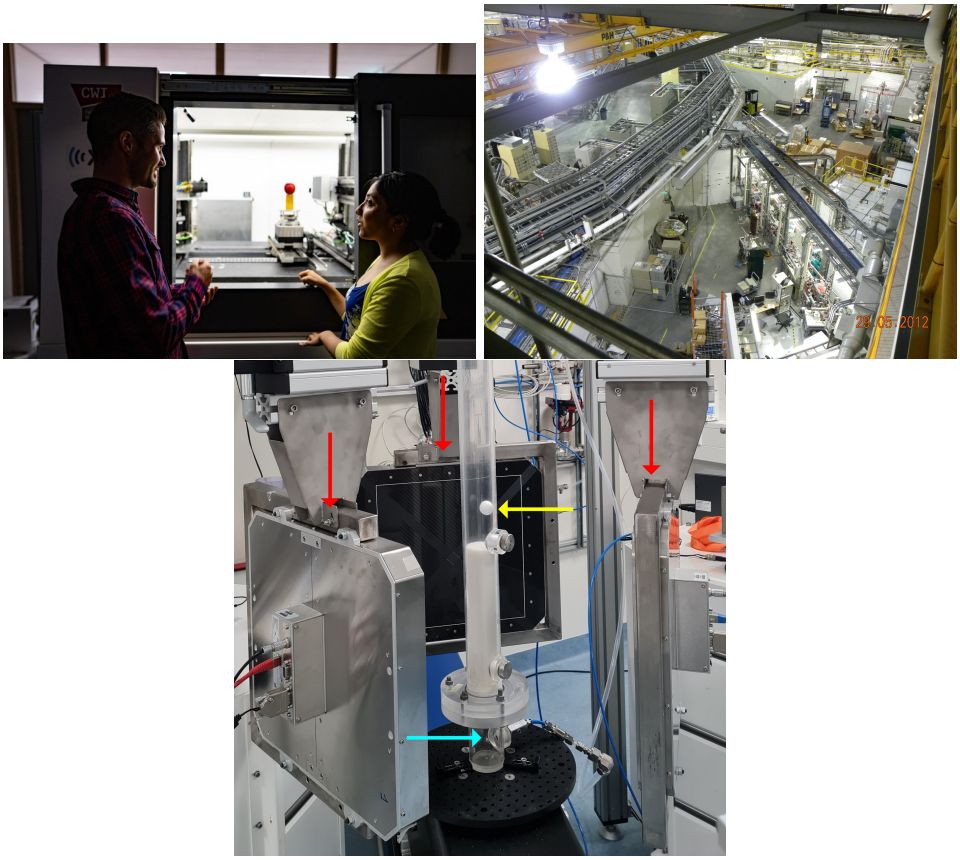


Figure 1.5: Upper left: Conebeam micro-CT scanner at the FleX-ray laboratory (CWI, Amsterdam, the Netherlands). **Upper right**: The SM beamline at the Canadian Light Source (CLS Research Office, Saskatoon, Canada), image provided under the CC BY-SA 2.0 license. **Bottom**: Multi-source multi-detector set-up (Delft University of Technology, Delft, the Netherlands).

storage ring to create brilliant, highly focused, monochromatic X-rays, thereby enabling a more precise investigation of samples than laboratory set-ups. Scientists from diverse fields of applied sciences, e.g., on materials and biological systems, frequently travel to synchrotrons to perform dynamic experiments. To help their users, synchrotrons conduct active research to store, process, and visualize data.

In some situations, the object dynamics can become too complicated or fast for the object to be adequately imaged using a system with a single source and detector. Modern medical devices, for instance, can perform a single rotation once every 0.25–0.30 seconds (3–4 Hz). This means that information on one particular angle of the object is only queried every 0.25–0.30 seconds, which can be too low of a temporal resolution to image certain dynamical processes. For these situations, systems with multiple sources and/or multiple detectors are considered, such as the system with three source-detector pairs in the bottom of figure 1.5. We will further explore the triplet source and detector set-up in Chapters 2 and 5.

Photon noise and electronic noise Similar to photographs taken under low-light conditions, radiographs require a sufficient number of photons to pass through the object and arrive at the X-ray detector. Thus, X-ray sources must emit a sufficient amount of photons, while detectors must register a sufficient amount of photons. Generating a high photon flux is a standing technical challenge. To see this, consider the CBCT set-up (figure 1.1). The X-ray tubes used in CBCT work by shooting electrons into a target material, such as tungsten. A small percentage of the electron energy is converted into X-rays, but the majority of energy dissipates into heat. Heat management is therefore an integral aspect of increasing the photon flux. Then again, to make use of higher photon flux, X-ray detectors must count and read-out the photons, which are converted to electric charge in the detector, at a faster pace. The detector design, such as flat-panel shape, sensor material, capacitance of the pixel sensors, and bandwidth of data buses like PCIe, all determine how fast the charge can be read from the sensor array.

Image quality depends on the number of photons captured and thus on the exposure time. In low-dose imaging, a grainy type of noise is observed known as *photon noise* or *shot noise*. This type of noise can be described by a Poisson probability distribution summarizing the X-ray photon generation in the X-ray source, the interactions with atoms in the sample, and the detection in the scintillator [3, 13, 14, 15]. A scintillator is a crystal, such as Caesium Iodine (CsI) or Gadolinium Oxysulfide (GadOx), consisting of high-atomic-number elements which increase the probability of photo-electric interaction. Detectors that use thick scintillator screens are able to capture more photons, but at the cost of stronger blurring of the measurement.

A second type of noise is due to the electronic components within the X-ray detector instrument. These noise characteristics depend mainly on the the type of photo-detectors used, for example, CMOS (Complementary Metal Oxide Semiconductor) active pixels. This type of noise is typically modeled as Gaussian. The combination of Poisson-Gaussian noise will be discussed more extensively in Chapter 5.

Sparse-view geometries The quality of a reconstruction depends on the spatial and angular sampling of the radiographs. For example, in Fourier-based reconstruction techniques, such as the FDK algorithm, the number of radiographs sampled in a circular trajectory determines until which frequency an object can be reconstructed [3]. For industrial applications (such as identifying knots in scanning of wood logs [16]), or scientific applications in synchrotrons [17], obtaining a sufficient number of angular samples often proves difficult.

Scientific applications use large-area flatpanels detectors that can cover larger parts of the to-be-scanned samples, but have a lower read-out speed compared to line detectors. As a result, detectors are often not fast enough to keep up with the dynamics of the object or could put constraints on the number of samples that can be scanned in high-throughput in-line tomography. The consequence is that the object or device must rotate faster and that fewer radiographs can be retrieved from a single rotation of the set-up. The reconstruction problem resulting from a sparse-angular range of data is known as *sparse-view CT*.

1.6 Faster reconstruction via tailored algorithms

Next to fast radiograph acquisition, tomographic pipelines require fast computational components to handle the extraordinarily large amounts of data that they pass. In the third set-up of figure 1.5, for example, a single timeframe of three detectors can contain the information to reconstruct the object on a 1500-by-1500-by-1500 spatial grid. With a framerate of 60 Hz and 64-bit numerical precision, the total size of all reconstructions after 20 minutes of experimentation would amount to 1,768 terabytes (1 terabyte is equivalent to 1,024 gigabytes or 1,048,576 megabytes). Efficient processing and reconstruction is then indispensable. This section discusses the role of GPU-accelerated tomographic projectors and their inclusion in software frameworks.

Tomographic projectors In Eq. 1.5, $\mathbf{A}\colon x \mapsto y$ (the X-ray transform) is often called a *forward projection* due to the geometric notion of the object casting a shadow on the detector. Similarly, its adjoint, $\mathbf{A}^T\colon y \mapsto x$, is called a *backprojection*. Several discretization strategies can be considered to construct $\mathbf{A}$ and $\mathbf{A}^T$ [4, 18]. Each row of $\mathbf{A}$ (which corresponds to a column of $\mathbf{A}^T$), discretizes a single line integral from Eq. 1.3. That is, it contains the interpolation weights for the integral

$$[\mathcal{A}x]_{u,v,\psi} = \int_{-\infty}^{\infty} x\left(s_\psi + (d_{\psi,u,v} - s_\psi)t\right)\,\mathrm{d}t, \tag{1.9}$$

which describes the straight line from a point source s_ψ to a detector pixel midpoint $d_{\psi,u,v}$ through the volume x.

A common choice for estimating the line integral (Eq. 1.9) is the Joseph kernel [4, 19]. In this *ray-driven* approach, each integration point takes a trilinearly interpolated value from neighboring points on the voxel grid. Importantly, the line is modeled such that it precisely arrives at a detector pixel's midpoint, and hence no sinogram interpolation is needed. Conversely, during a *voxel-driven* backprojection, a bilinear interpolation at each

angle of the sinogram sums up to the voxel's value [20]. In this case, all lines of backprojection go precisely through the voxel's center, and now, interpolation in the volume is avoided. The reader is referred to [4] for interpolation formulae.

Parallel implementation on GPUs Most numerical implementations of projectors nowadays rely on hardware acceleration using graphical processing units (GPUs) (Chapter 4). GPUs consist of a large number of computational cores, and are therefore generally more efficient for large-scale parallel operations than a computer's central processing unit (CPU). The most common platform for writing GPU-accelerated code today is Nvidia CUDA. Listing 1.1 demonstrates a purposefully-simplified piece of CUDA code that performs backprojection for a circular fan-beam geometry. The fan-beam geometry is equivalent to a two-dimensional cone-beam geometry in the horizontal plane.

Listing 1.1: A simplified CUDA kernel for fan-beam backprojection used for demonstration on github.com/adriaangraas/astra-kernelkit. This function is launched in parallel with one thread per voxel (X, Y) in the volume.

```
__global__ void backproject(
   float * volume, float * sinogram, float * angles,
   int nr_angles, int nr_voxels_x, int nr_voxels_y, int nr_pixels,
   float src_obj_dist, float obj_det_dist
) {
   // X, Y are the voxel's center coordinates of this thread
   unsigned x = blockIdx.x * blockDim.x + threadIdx.x;
   unsigned y = blockIdx.y * blockDim.y + threadIdx.y;
   float X = float(x) - nr_voxels_x / 2.0 + 0.5;
   float Y = float(y) - nr_voxels_y / 2.0 + 0.5;

   // if this thread number does not map to a voxel, just ignore it
   if (x >= nr_voxels_x || y >= nr_voxels_y) return;

   for (int psi = 0; psi < nr_angles; ++psi) {
      // project the voxel (X, Y) onto a coordinate U on the 1D detector
      float U = forward_project(
         X, Y, angles[psi], src_obj_dist, obj_det_dist);
      // linearly interpolate the value at U from sinogram gridpoints
      volume[y * nr_voxels_x + x]
         += linearly_interpolate(sinogram, psi, nr_pixels, U);
   }
}
```

In this CUDA function, termed a *kernel* in the jargon of CUDA computing, the arrays `float * volume` and `float * sinogram` are passed as pointer arguments. This means no memory copies are made, and implies that the values reside in global GPU memory. The parameters of the call, furthermore, contain the geometry description of the 2D reconstruction. The kernel is launched in parallel, using a block of *threads* organized by pairs of 2D indices.

Each thread (`x`, `y`) maps to a single voxel coordinate (`X`, `Y`), and computes and writes its outcome to `volume`. In the function, a loop accumulates the contribution of each angle, by first projecting the voxel onto the detector at the specified angle, and then bilinearly interpolation using the nearest pixel values. CUDA kernels thus enable a matrix-free approach of computing the volume from the sinogram, i.e., the backprojection matrix $\mathbf{A}^T$ was never explicitly formed in memory.

The combination of a ray-driven forward projector and voxel-driven backprojector has advantages for an implementation on GPUs. All threads (discussed in Chapter 4) are independent of each other, which avoids potential race conditions, i.e., when two threads would write to the same memory simultaneously [7]. However, due to the difference in forward and backward lines, the projectors are not each others exact transpose (this is called *unmatched*). Unmatched projectors lead to nonconvergence in iterative algorithms, due to a nonsymmetry of the iteration matrix [21]. In the presence of noise, this does not always pose a problem [22].

Software implementations The precise implementation of algorithms in tomographic pipelines can play a critical role in their efficiency. Common software packages used by researchers are the ASTRA Toolbox [23], TIGRE [24], or packages that use these frameworks as a back-end, such as TomoPy [25], Tomosipo [26] or the MATLAB Spot toolbox. Via several software layers (e.g., Python, MATLAB, Cython, C++), these packages call CUDA kernels similar to Listing 1.1 to implement algorithms such as SIRT and FBP efficiently.

For sequential tomographic reconstructions in pipelines, standard packages can quickly become inefficient, as they are geared towards making single reconstructions. To see this, compare Listing 1.2 with Listing 1.3, which are pseudo-codes for a repeated filtered-backprojection reconstruction from a sliding window of projections. The left listing treats each reconstruction individually: re-uploading measurement data, recomputing the same geometry, and repeating a filtering step with the Ram-Lak filter unnecessarily. The implementation in the right listing instead removes transfers and geometry computations from the sequential loop, thereby reducing bandwidth and computations.

Listing 1.2: Naive FBP implementation for a sliding projection window.

```
for t in [0, 1, ..., T - 360]:
   # repeat reconstruction
   geom = compute_geometry(
     src, det, angles)
   y = upload_to_GPU(  # allocate
     [data[t], ..., data[t+360]])
   y = convolution(y, ram_lak)
   x = backproject(y, geom)
   delete y, x  # deallocate
```

Listing 1.3: Incremental FBP implementation, recycling memory and computations.

```
geom = compute_geometry(
   src, det, angles)
y = upload_to_GPU(  # allocate once
   [data[0], ..., data[T-1]])

for t in [0, 1, ..., T - 360]:
   # update a single angle
   y[t % 360] = convolution(
      y[t], ram_lak)
   x = backproject(y, geom)

delete y, x  # deallocate
```

While the example above is purposefully simplified, many current software packages do not permit the flexibility of the second form, Listing 1.3, which requires a more fine-grained control over, e.g., GPU memory. The reason is often that an implementation is hidden by language barriers in the software. Sometimes, users of tomographic packages find inventive ways to use existing software more efficiently, for example, by reformulating a stack of 2D reconstructions as if the data were acquired from a single 3D geometry, using the z-dimension for the stack. In Chapter 4 we will further discuss how software packages can become more flexible by eliminating the barrier between the high-level "prototyping language" and the low-level "efficient language".

Tuning reconstruction problems In tomographic CUDA kernels, such as Listing 1.1, there are often several free parameters: the number of angles handled by one parallel thread, the distribution of voxels over threads, as well as certain numerical choices such as the axis ordering of the geometry, interpolation method, and types of memory to store data and intermediate computations. Oftentimes, sensible defaults already yield good performance, however, for unconventional reconstruction problems the standard choices can be significantly less efficient.

Kernel tuning aims at optimizing free parameters in kernels. Tuned algorithms can achieve better run-times, reduce energy consumption [27, 28], or utilize less resources, in particular, GPU memory and computation. In high-throughput applications, such as in-line CT scanning, a kernel can be tuned toward a fixed measurement protocol and dedicated GPU architecture. In these situations, even a slight improvement can lead to significant energy savings over the equipment's lifetime.

1.7 Introduction into Deep Learning

For inverse problems in imaging, it would be no exaggeration to call our last decade the *decade of deep learning*. Deep learning is a form of Machine Learning (ML) or Artificial Intelligence (AI) using neural networks consisting of many consecutive operations (*layers*). It brought a new perspective on algorithm design, showing that existing algorithms can be replaced or improved by "learned" alternatives, which means that these algorithms have free parameters that are initialized based on prior information obtained from (large) data sets of examples.

In tomography, deep learning-based algorithms are currently considered the state-of-the-art [29, 30, 31, 32]. Moreover, —and this is an often under-emphasized advantage— neural networks can be much faster than traditional algorithms, especially when they are executed on GPUs. Compared to traditional algorithms, neural networks can perform certain image enhancement tasks in milliseconds to seconds, rather than in minutes or hours. This allows supersampling, artifact removal, segmentation, or denoising [33, 34, 35] to be executed at high framerates in tomography pipelines, especially in synchrotron and laboratory environments. The disadvantage of deep learning, however, is that these algorithms need to be trained before usage, which can take days to weeks to complete.

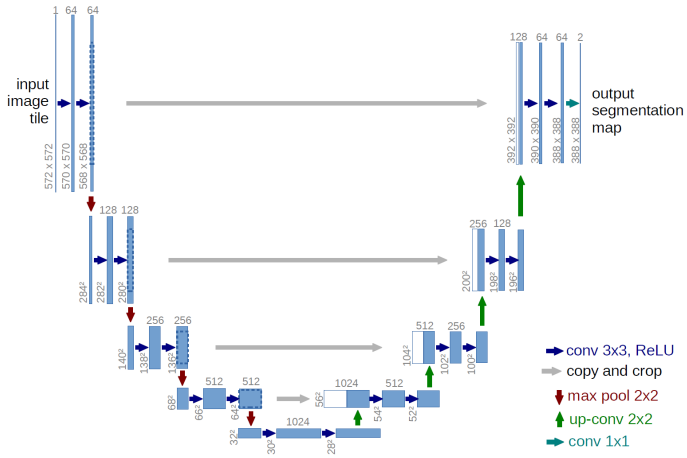


Figure 1.6: The U-Net architecture [38] is a DNN consisting of an encoding-decoding architecture using many convolutional layers, shaped in a "U". The implementation illustrated here relies on learnable "max pooling" and "up-convolution" operators to decrease and increase the internal image resolution.

Next, the three pillars of deep learning are explained: *representation*, *optimization* and *generalization*.

Representation Designing a deep neural network (DNN) architecture that is well suited to the task at hand, is the area of *representation*. A design concerns the number of layers and channels, choices of operators, and connections in the network. For imaging tasks, architectures often consist of convolutional layers and non-linear activation functions [36, 37]. One prominent example is the U-Net (figure 1.6), a neural network that downsamples and upsamples the internal representation of the image, in order to detect image features on multiple scales.

The goal of a neural network architecture f is to approximate an unknown mapping $f^\dagger : \mathcal{U} \to \mathcal{V}$, where $\mathcal{U}$ and $\mathcal{V}$ denote image manifolds in case of an image-to-image network. For image denoising, for example, the aim of f would be to approximate the perfect denoiser $f^\dagger$. In this case, the ideal but unknown $f^\dagger$ would map each noisy input image from $\mathcal{U}$ to a denoised counterpart in $\mathcal{V}$. Another example of an image-to-image neural network would be one that removes sparse-angle artifacts from reconstructions (cf. figure 1.2 and figure 1.3).

The approximation f of $f^\dagger$ is accomplished by a parametrization, which is denoted by $f \equiv f_\theta$, with $\theta \in \Theta$ being the free parameters. Those parameters can be, for example, the filters of convolution operators or the matrix entries of linear operators.

Optimization During *optimization*, or *training*, the parameters θ are obtained using a data set of input-target examples. Let $\mathbf{u}_i \in \mathcal{U}$ denote an input—for example, a 2D or 3D noisy reconstruction—and $\mathbf{v}_i \in \mathcal{V}$ denote a target. In *supervised learning*, the target must be a

ground truth, i.e., $\mathbf{v}_i \approx f^\dagger(\mathbf{u}_i)$. In the case of supervised denoising, for example, the input could be a noisy reconstruction, and the target must then be a noise-free counterpart.

Given a task-specific *loss function* $\ell(\mathbf{u}, \mathbf{v})$ that measures the misfit between two images, training can be formulated as the optimization of the so-called *empirical risk* $R_\mathcal{D}$ for parameters θ, data set $\mathcal{D}$, architecture f_θ and loss ℓ:

$$\min_{\theta \in \Theta} \quad R_\mathcal{D}[f_\theta] := \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{u}_i, \mathbf{v}_i) \in \mathcal{D}} \ell\left(f_\theta(\mathbf{u}_i), \mathbf{v}_i\right). \tag{1.10}$$

Generalization The performance of a trained network on unseen data is called *generalization*. To quantify generalization, image pairs from $\mathcal{D}$ are considered as random samples from a latent *training distribution*, a probability distribution $\mathbb{P}$ over $\mathcal{U} \times \mathcal{V}$. The *expected risk* describes the true but unknown performance of f_θ over $\mathbb{P}$,

$$R_\mathbb{P}[f_\theta] := \mathbb{E}_{(\mathbf{u},\mathbf{v})\sim\mathbb{P}}\left[\ell(f_\theta(\mathbf{u}), \mathbf{v})\right], \tag{1.11}$$

whereas the empirical risk is limited to a finite sample of it. The *generalization gap*, $|R_\mathcal{D} - R_\mathbb{P}|$, describes the distance between the two risks, and a network is said to generalize well when this gap is small. Since the expected risk cannot be computed, $R_\mathbb{P}$ is commonly approximated using a *hold-out data set* $\mathcal{D}'$ also sampled from $\mathbb{P}$ (also often referred to as the *test data set*). One then computes $|R_\mathcal{D} - R_{\mathcal{D}'}|$ for an approximation of the generalization gap.

1.8 Self-supervised Learning: Scan faster, repair later

Learned algorithms have the potential to remove noise and sparse-angle artifacts that were introduced by fast scanning procedures (section 1.5), and thereby allow to image with higher temporal resolution or throughput. In Chapters 3 and 4 we will discuss several challenges that these algorithms bring, such as a sufficient amount of training time or integrated tomographic projectors. Their largest challenge, however, is typically the availability of ground truth training data. Low-noise or artifact-free images are difficult to acquire in real-world applications, for example, because of limitations of the equipment, laboratory time, or during scanning of fast dynamic processes. At the same time, learned tomographic algorithms cannot rely on the large training data sets that are publicly available on the internet. These data sets do not generalize well, as experimental tomographic data consists of unique image features.

To overcome these difficulties, *self-supervised learning* replaces the ground truth targets $\mathbf{v}_i$ in the data set by noisy surrogates that are easier to obtain in practice. Even though $\mathbf{v}_i \not\approx f^\dagger(\mathbf{u}_i)$, training with surrogates can still yield a network f_θ that approximates the sought $f^\dagger$ nonetheless. In the remainder of this section we will give examples of self-supervised denoising.

The *Noise2Noise* strategy: assume paired images The *Noise2Noise* [39] (N2N) strategy is one of the most influential works published on self-supervised denoising. N2N requires a training data set $\mathcal{D}$ consisting of pairs $(\mathbf{u}_i', \mathbf{u}_i'')$, where $\mathbf{u}_i'$ and $\mathbf{u}_i''$ are two images of the same object but with different i.i.d. (independent and identically distributed) realizations of the latent noise distribution. Similar to the supervised loss (equation 1.10), the goal is to minimize

$$\min_{\theta \in \Theta} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{u}_i', \mathbf{u}_i'') \in \mathcal{D}} \ell\left(f_\theta(\mathbf{u}_i'), \mathbf{u}_i''\right). \tag{1.12}$$

Equation 1.12 thus simply takes a noisy target $\mathbf{u}''$ instead of the ground truth $\mathbf{v}$. Since the noise is not correlated between inputs and targets, f_θ can at best reproduce noiseless image features that are consistent between inputs and targets. Those image features are encoded as convolutional filters in θ, and are learned due to the similarity of patterns in the data set.

The main limitation of Noise2Noise is that it requires paired acquisition, i.e., two noisy images are required for the network to be trained. The next two examples show two more self-supervised denoising strategies that are derived from Noise2Noise and bypass this requirement.

The *Noise2Inverse* strategy: splitting the angular projections Noise2Noise is brought to the tomographic domain by *Noise2Inverse* [40] (N2I). In its simplest form, it chooses for $\mathbf{u}'$ an FBP reconstruction from projection angles with even indices and for $\mathbf{u}''$ an FBP reconstruction from odd indices. Assuming that the detector noise is temporally uncorrelated, $\mathbf{u}'$ and $\mathbf{u}''$ result in closely similar objects, but with statistically independent noise. It thereby avoids acquisition with two noisy projections for each angle.

In Chapter 3 we will discuss a real-time pipeline built with Noise2Inverse:

FleX-ray set-up → Region-of-interest FBP → *Noise2Inverse*	(1.13)

Here, N2I is executed as a post-processing neural network to denoise filtered-backprojection reconstructions. To keep the pipeline fast, training the reconstruction algorithm is executed on patches, i.e., small regions-of-interest, meaning that the size of the reconstruction is restricted.

The *Noise2Self* strategy: splitting by pixels Blind-spot denoisers are another type of self-supervised methods that aim to overcome the problem of paired noisy measurements. The method is not restricted to the tomographic domain but can be applied to any data set of images as long as assumptions on the noise are satisfied. Blind-spot denoisers are based on the idea that the image features within a single image have spatially-extended structures, whereas pixel-wise noise is statistically independent. The seminal works are Noise2Self [41] and Noise2Void [42], to which there have been several extensions and improvements [43, 44, 45]. A formalization of their principle is termed $\mathcal{J}$-invariance [41], which describes the statistical independence of a function (in our case, a neural network) between subsets of its

input and its output. In practice, it is enforced by masking or randomization of pixels in the input. Definition 1.8.1 gives our tailored redefinition of $\mathcal{J}$-invariance for images.

Definition 1.8.1 ($\mathcal{J}$-invariance)**.** *Let $\mathcal{J}$ be a partition of the pixel indices $\{0, \ldots, N_u N_v\}$ into non-overlapping grids. Let $J \in \mathcal{J}$ be one grid, and denote with $(\cdot)_J$ its values. An $f_\theta \colon \mathbb{R}^{N_u N_v} \to \mathbb{R}^{N_u N_v}$ is called* $\mathcal{J}$-invariant *if $\mathbf{u}_J$ and $(f_\theta(\mathbf{u}))_J$ are statistically independent for all $J \in \mathcal{J}$.*

To understand why $\mathcal{J}$-invariance is effective, recall that in a self-supervised method the ground truth target $\mathbf{v}$ is replaced by a noisy realization. Blind-spot denoisers choose for this realization the input $\mathbf{u}$ again and constrain f_θ to be a $\mathcal{J}$-invariant function. The expansion of a self-supervised mean-squared error loss, with $\mathbf{u} = \mathbf{v} + (\mathbf{u} - \mathbf{v})$ decomposed in ground truth and noise, gives:

$$\mathbb{E}\,\|f_\theta(\mathbf{u}) - \mathbf{u}\|_2^2 = \mathbb{E}\|f_\theta(\mathbf{u}) - \mathbf{v}\|_2^2 + \mathbb{E}\,\|\mathbf{u} - \mathbf{v}\|_2^2 - 2\mathbb{E}\,\langle f_\theta(\mathbf{u}) - \mathbf{v}, \mathbf{u} - \mathbf{v}\rangle\,, \tag{1.14}$$

which consists of the supervised loss, but adds two more terms: A variance term and a negative cross-product. The variance term does not depend on f_θ, and therefore does not play a direct role during training. The cross-product describes the correlation between noise in the input (i.e., the right-hand term, $\mathbf{u}-\mathbf{v}$) and noise in the output (i.e., the left-hand term, $f_\theta(\mathbf{u})-\mathbf{v}$). In a $\mathcal{J}$-invariant network, by definition 1.8.1, $f_\theta(\mathbf{u})-\mathbf{v}$ and $\mathbf{u}-\mathbf{v}$ do not correlate, and the cross-term therefore vanishes, regardless of the choice of network parameters θ. Therefore, optimization of a blind-spot network resembles supervised optimization. Note, however, that constraining f_θ to be a $\mathcal{J}$-invariant network will typically lower its ability to represent $f^\dagger$, e.g., it may perform less well at denoising compared to an unconstrained network trained in a supervised way.

In Chapter 5, we will consider the tomographic pipeline

$$\boxed{\text{TU Delft set-up} \to \textit{Noise2Self} \to \text{SIRT reconstruction.}} \tag{1.15}$$

In comparison to the previous pipeline, the learned component appears in the middle, and removes noise *before* data enters the reconstruction algorithm.

1.9 Research Questions

In this dissertation, several aspects of fast CT have come together in four research questions, each associated with a chapter and a journal publication. Although unintentional, the chronology of the publications roughly outline that of a "fast tomographic pipeline": The first work is on the topic of fast acquisition, the second on real-time imaging pipelines, the third on fast reconstruction software, and the last paper on self-supervised deep learning.

The first paper publication is a collaboration with Delft University of Technology and in particular with Evert Wagner and Luis Portela from the Transport Phenomena group. It describes an X-ray set-up dedicated to imaging the fast dynamics of fluidized beds with

a tailored reconstruction method. The project entailed several challenges with real-world data, such as the set-up calibration, and the development of a robust X-ray measurement principle. The collaboration also provided a data set that was used in the third and fourth research projects.

The second project discusses *just-in-time deep learning*, a learning technique for real-time tomographic pipelines. The original motivation for the work stemmed from the RECAST3D software project, developed by Jan-Willem Buurlage and other members of the Computational Imaging group at CWI. RECAST3D facilitates real-time reconstruction by providing a graphical user interface around FDK/FBP reconstruction restricted to cross-sectional slices. Since high-resolution real-time 3D reconstruction is not yet feasible, the software aims to reconstruct only what the user needs to see, effectively reducing the 3D problem to a few 2D problems. The existing pipeline opened the opportunity to explore real-time learning as well as a real-time reconstruct-and-denoise set-up. The data used to demonstrate the concept was acquired in the FleX-ray laboratory at CWI by Felix Lucka and Sophia Bethany Coban.

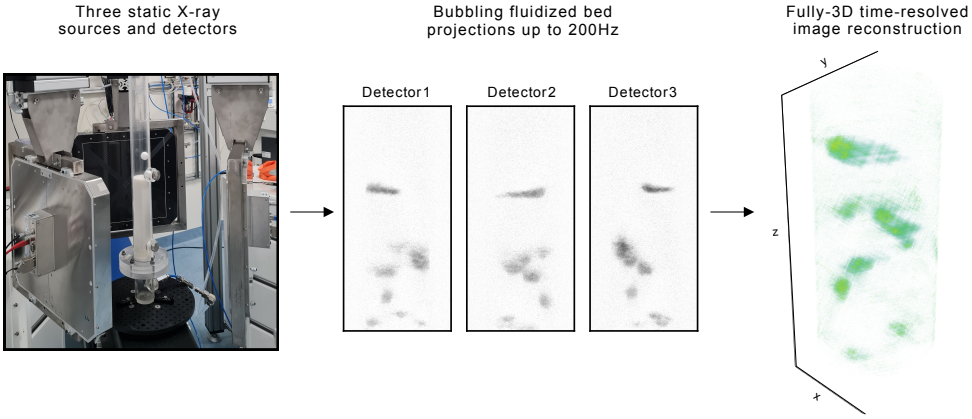
The third project concerns a software project that enables faster reconstruction for unconventional geometries or 4D reconstruction problems. The software is a continuation of the first two projects. The original reconstruction algorithm used by TU Delft required a CPU operation inside of a GPU algorithm, resulting in excessive algorithm runtimes. The second project then required thousands of reconstructions per second for the purpose of on-the-fly training of neural networks. Both of these scenarios could not be optimally addressed with the reconstruction software that was present at the time. The paper was written together with Willem Jan Palenstijn and Ben van Werkhoven from Leiden University.

The focus of the last research question and paper publication was on denoising, an important topic within the first and second project. In tomographic imaging of fast dynamic processes, radiographs can degrade severely due to noise. Although several techniques exist to remove noise in image space (after reconstruction), denoising radiographs has particular advantages, especially with sparse or ultra-sparse geometries. Hence, a self-supervised deep-learned denoising method based on Noise2Self was investigated that could work entirely without ground truth or noisy training targets.

1

Can gas-solids fluidized beds be imaged with Computed Tomography, without sacrificing a spatial or temporal dimension? (Chapter 2)

A fluidized bed is a dynamic gas-particle mixture that has fluid-like characteristics. Due to its fast dynamics, it can only be imaged with a set-up consisting of multiple non-rotating sources and detectors. Current imaging methods either use a high-speed slice-based (2D) reconstruction, on a single height of the bed, or a temporally-averaged 3D reconstruction, over the bed's full height. In this chapter, we present a method that is both fully-3D and time-resolved. It uses a set-up with three source-detector pairs, a marker-based calibration procedure, a tailored preprocessing technique, and a constrained SIRT reconstruction. In comparison to existing techniques and other modalities, the X-ray technique enables investigation of the high-velocity morphological changes and interactions of bubbles in large-scale laboratory simulations of fluidized beds.



How to overcome the issue of neural network generalization in real-time tomography? (Chapter 3)

Many synchrotron tomography beamlines and X-ray laboratories are moving toward real-time visualization and experimental steering using tomographic imaging pipelines. Deep learning has an incredible potential for these pipelines, e.g., as denoising or segmentation add-ons, as it can provide high framerates and is typically more accurate than traditional methods. However, due to distributional changes in the data during experiments, e.g., due to user interactions, experimental modifications, and set-up reconfigurations, there is generally no reliable training data available, and therefore networks do not generalize well. We propose to train small CNNs concurrently with the ongoing experiment by intercepting the reconstructions that are pushed through the pipeline. In our experiments from the FleX-ray laboratory, we show that a denoising network generalizes better than a pre-trained CNN on pipeline data of dissolving tablets with RECAST3D.

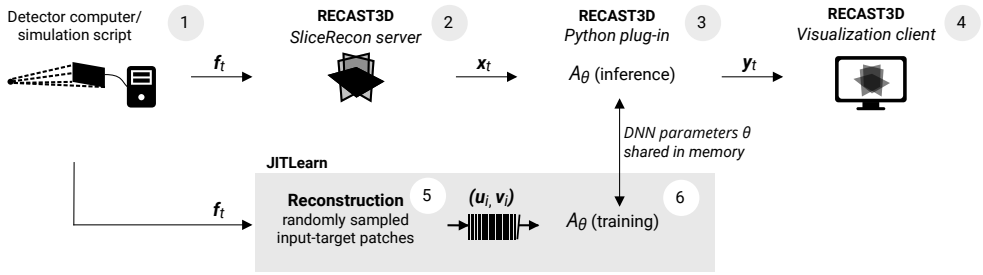


Figure 1.7: Just-in-time learning using a U-Net for image denoising. **Labels 1 to 4**: The pipeline reconstructs slices $\mathbf{x}_t$ from a stream of projections $\mathbf{f}_t$, and sends these to the visualization client. **Labels 5 and 6**: The neural network process intercepts the slices via a plug-in at (3). In a concurrent process, the U-Net architecture A_θ is trained on reconstructions using a capacity queue.

1

How to make the high-performance ASTRA Toolbox projectors easily modifiable for Python end-users? (Chapter 4)

Tomographic algorithms often use matrix-free implementations of the X-ray operators $\mathbf{A}$ and $\mathbf{A}^T$ (equation 1.5) called *forward* and *backprojectors*. In ASTRA Toolbox and Tomosipo, they are optimized for a generic geometry and data size, which can lead to suboptimal performance in neural networks, with very small and high-volume data, or in dynamic reconstructions. Modifying or tuning them towards a specific use case, however, is not straightforward due to several programming layers of abstractions between the user's programming language (Python/MATLAB) and the graphics processing unit (GPU). In this article, we use CuPy, a Python library resembling NumPy and SciPy, to compile GPU projectors during the Python script. We then show improved performance for problems with non-standard geometries and data sizes.

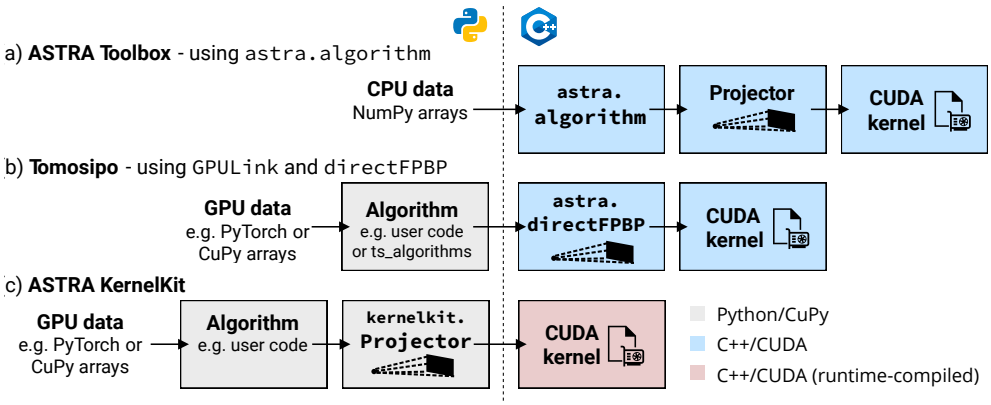


Figure 1.8: Software overview of ASTRA Toolbox, Tomosipo [26], and our package, ASTRA KernelKit, illustrating where its components are located with respect to the Python-C++ language barrier. (a) and (b) show two methods of accessing a projector in ASTRA Toolbox. In comparison, ASTRA KernelKit uses a Python-based projector and runtime compilation of the CUDA kernel.

How to denoise radiographs using neural networks, without ground truth example data? (Chapter 5)

Recently, a new class of self-supervised denoising strategies has emerged for data sets consisting of *unpaired* noisy images, i.e., without clean ground truths or even secondary noisy realizations. Despite their large potential for X-ray imaging, where dynamical set-ups often acquire long sequences of single, noisy, X-ray radiographs, these methods cannot be applied straightforwardly. The main problem is that X-ray scintillator detectors cause a spatial correlation of the noise, preventing denoisers in this class from distinguishing clean image features (which correlate spatially) from noise (which does not correlate spatially). We show that, because the point-response function of scintillator blur is highly uniform, the noise correlation can be reverted by a deconvolution. This allows self-supervised denoising of the preprocessed radiographs while leveraging image features from large experimental data sets.

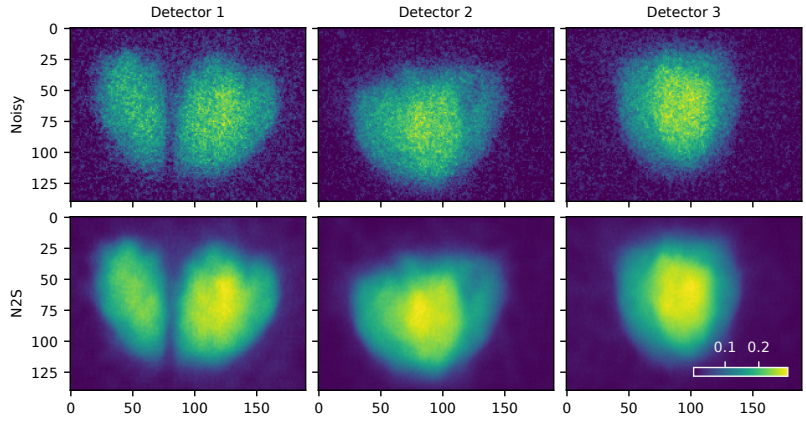


Figure 1.9: An application of the trained self-supervised denoiser Noise2Self (N2S) to a sample of a large data set of noisy fluidized-beds radiographs without ground truths.

1

Chapter 2

X-ray Tomography for Fully-3D Time-Resolved Reconstruction of Bubbling Fluidized Beds

This chapter is based on the article: Adriaan B.M. Graas, Evert C. Wagner, Tristan van Leeuwen, J. Ruud van Ommen, K. Joost Batenburg, Felix Lucka, and Luis M. Portela. "X-ray tomography for fully-3D time-resolved reconstruction of bubbling fluidized beds". In: *Powder Technology* 434 (2024), p. 119269. ISSN: 0032-5910. DOI: 10.1016/j.powtec.2023.119269

2.1 Introduction

In gas-solid fluidized beds, a mixture of gas and solid particles attains fluid-like characteristics. Fluidized beds are widely applied in chemical, pharmaceutical, and mineral industries, and laboratory-scale set-ups are used to investigate and validate fluidized beds in production applications. Gas is typically added from the bottom of a column or tank, and the *solids*, i.e., suspended particles, achieve different mixing regimes depending on particle size and gas velocity [47]. In bubbling regimes, a gas-solid fluidized bed has a homogeneous dense phase, and gas travels in dispersed voids, i.e., *bubbles*, upwards through the bed.

Experimental imaging of the gas-solid distribution of bubbling beds traditionally aims to find quantities such as sizes and shapes of bubbles, as well as the bubbles' solid contents. These are of paramount importance, e.g., to explain the catalytic behavior, or to describe the conversion of reactants. Since the solids mixture is opaque to visible light, several alternatives to optical imaging have been developed in the past decades [48]. Intrusive techniques, such as optical probes, use point measurements to arrive at local gas holdup, cord length and bubble velocity [49, 50]. Non-intrusive techniques, such as electrical capacitance tomography (ECT) [51], and X-ray computed tomography (CT), resolve the spatial distribution of the solids in a 2D slice or 3D volume. Positron emission particle tracking (PEPT) [52] measures the particle velocity in a volume. Of a more recent interest is also the study of the dynamic behavior of bubbles and solids. This, however, requires a technique for a fully spatio-temporally resolved gas-solid distribution. Bubble motion has a strong effect on the convection of particles, and, therefore, plays an important role in mass transport and advection of heat.

At Delft University of Technology, a set-up consisting of a triplet of conebeam sources (X-ray tubes) and 32-by-2 double-line detectors was introduced in 2010 [53]. The three sources and detectors were placed in an equilateral triangular geometry. This enabled tomographic reconstruction of bubbles in the horizontal plane of a 24 cm diameter column. Thanks to the double-line detectors, which operated at an effective 250 Hz, bubble velocities could be inferred. This subsequently allowed pseudo-3D bubble reconstruction through stacking the temporal evolution of the 2D slice [54]. Recently, the detectors have been upgraded to 1548-by-1524 pixel flat panels, and have enabled X-ray radiographic experiments to measure gas holdup in fluidized beds, cavitation, and bubble columns [55, 56, 57]. As our new detectors have the ability to capture the complete 3D volume in every frame, we are a step closer towards tomographic reconstructions that are both fully-3D and time-resolved.

In this work, we introduce and evaluate our new experimental method for tomographic reconstruction. This consists of (i) the stationary set-up, with three sources and detectors, (ii) a geometric calibration procedure and data processing method, and (iii) the Simultaneous Iterative Reconstruction Technique (SIRT). We examine its limitations, and present its capabilities. Since there exists a large amount of related techniques on tomography of multiphase flows, including fluidized beds, we start by giving a brief overview of work related to ours in Section 2.2. In Section 2.3, we introduce the set-up and calibration procedure. In Section 2.4, we describe a data processing method that is necessary to directly obtain tomographic reconstructions from the gas-solid distribution, and introduce SIRT. In Section 2.5,

we study reconstruction artifacts with numerical simulations, perform phantom experiments, and demonstrate its application in the acquisition of time-resolved reconstructions from Geldart B bubbling fluidized beds. In Section 2.6, we conclude with a reflection on the possibilities and limitations of the new technique.

2.2 Related work

X-ray set-ups for imaging dynamic processes are common in engineering, chemical and medical sciences [58]. Single-source systems allow acquisition with a fast-rotating component, and may use a 4D (i.e., 3D + time) motion-corrected or sparse-angular technique for reconstruction. When measurements must be acquired in a shorter time than what is achievable by a gantry, often referred to as *ultrafast* imaging, multi-source multi-detector systems are considered [59, 60]. Example applications are combustion processes, multiphase pipe flows, and cardiography, and set-ups range from a dual source and detector pair on a gantry [61] to 29 pairs on a stationary circular track [62].

For the purpose of imaging dynamics in fluidized beds several techniques have been proposed. Multi-source X-ray set-ups, on the basis of line detectors are an established technique [53, 59]. In early gamma-ray CT and radioactive particle tracking [52], for example, this enabled a time-averaged gas distribution or time-averaged velocity field [63, 64]. In recent decades, several experimental methods have been developed to image dynamics in fully-3D [65], although for some techniques this limited the temporal resolution. Slow-moving solids, for instance, can be analyzed in 3D using X-ray radiography with Particle Image Velocimetry (PIV) [66]. To our knowledge, currently three set-ups have been used for fully-3D time-resolved tomography of fluidized beds: (i) an electron beam X-ray CT scanner at Helmholtz-Zentrum Dresden-Rossendorf, achieves a high spatial resolution with a 500 Hz temporal resolution in an 8 mm thick cross-sectional volume [67]; (ii) a fully-3D time-resolved MRI technique has been demonstrated with single bubble injection at 140 Hz [68]; (iii) electrical capacitance volume tomography (ECVT) has been applied in a $20\times20\times20$-sized volume of size 5 cm width and 10 cm height at 80 Hz [69].

X-ray is a useful modality for imaging large scale multiphase flows in process engineering sciences, although it requires an environment for working with ionizing radiation [3, 5]. Magnetic Resonance Imaging (MRI) is a costly alternative, and can image particles that contain MR-sensitive nuclei in small volumes [70]. It delivers good contrast and recently achieved good temporal resolution [71]. Other modalities that have been successfully employed in 3D are the electrical resistance and capacity tomographic modalities, ERT and ECT/ECVT. They are inexpensive soft-field techniques that are able to reach a very high temporal resolution [72]. However, increasing their low spatial resolution is difficult due the limitations imposed by the placement of additional electrode pairs [51, 65, 70], and the unfavourable properties of the underlying mathematical image reconstruction problem [73]. The reader is referred to [65] for an overview and comparison of modalities.

In comparison to the existing 4D techniques, the main advantage of our set-up is that it enables imaging large scale flows at all heights simultaneously, with a high vertical resolution

and at a high framerate. It allows following individual bubbles along their entire paths upwards through a fluidized bed, and it therefore does not require statistical quantities for the analysis of bubble dynamics. While the readout bottleneck of CMOS detectors limits the attainable framerates, the most prominent limitation of our set-up is the effective resolution in the horizontal plane, which is surpassed by the previously mentioned electron-beam and MRI set-ups. Its potential resolution is 1548-by-1548 voxels, the number of pixels in a detector row, but the associated reconstruction problem is exceptionally difficult to solve: all n^2 voxels in the n-by-n horizontal plane must be resolved from only $\sim 3n$ values associated with their X-ray projections on the detectors. While the spatial localization of measured X-rays is very precise, compared to the ERT/ECVT and magnetic modalities, the quality of our set-up depends on the ability of the reconstruction algorithm to explain the measurement data. This ability will be tested experimentally in the forthcoming sections, and we will reflect on the development of new reconstruction algorithms in the conclusion.

2.3 Material and methods

2.3.1 Experimental set-up

Our set-up, aimed at tomographic reconstruction, consists of three continuous X-ray sources and three CMOS detectors, arranged in an equilateral triangle (see Fig. 2.1). Fig. 2.2 shows a photograph of the set-up with a polymethyl methacrylate (PMMA) cylinder in the center. A rotation table is used for calibration, and air is supplied at the bottom of the cylinder through a bundle of needles. The sources and detectors can be repositioned to accommodate the experiment. Detectors are synchronized using an external trigger. With the detectors configured to operate in a region of interest (ROI), i.e., by disabling a segment of the detector rows, framerates up to 200 Hz at 1548×100 pixels can be achieved. Before preprocessing (Section 2.4.1), we subtracted darkfield images, i.e., measurement frames with the sources off, and performed dead-pixel corrections on our measurements [74]. Dead pixels are defected pixels that are (partially) non-responsive to X-ray radiation, and appear along certain detector rows and columns in our data. They are filled in by an unweighted average over their direct non-defected neighbours.

For the experiments in Section 2.5, we placed the detectors relatively close to the column, and image about 20 cm of bed height. We cropped to 1548×550 pixels, which resulted in a framerate of about 65 Hz, or, equivalently, an exposure time of 15 ms. The PMMA cylinder contained spherical polystyrene particles with a diameter of 560 µm (Geldart B type) [53]. The minimum fluidization velocity of this material has experimentally been determined to be 15 cm/s. Set-up specifications and further details are summarized in Table 2.1.

2.3.2 Calibration

For an accurate tomographic reconstruction (cf. Section 1.1), a precise and coherent prescription of the geometry is essential. The geometry consists of all positions of sources and detectors, as well as the orientation of each detector plane, in a single coordinate system. Per source-detector pair, this can be encoded in nine parameters: two 3D positions and three

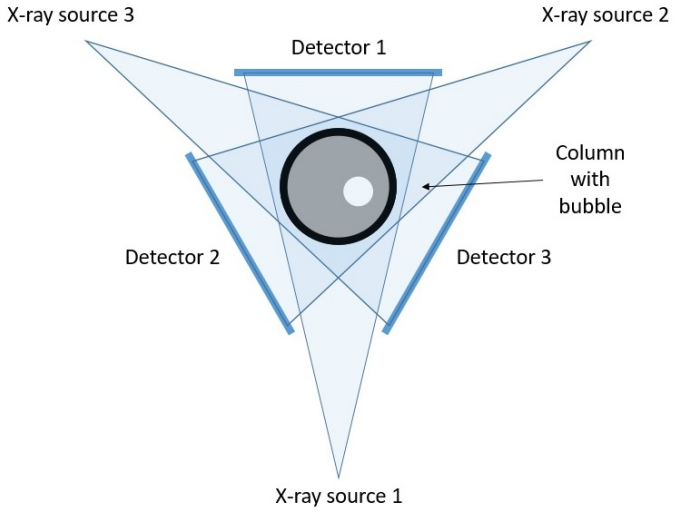


Figure 2.1: Top view schematic of the X-ray set-up (not to scale).

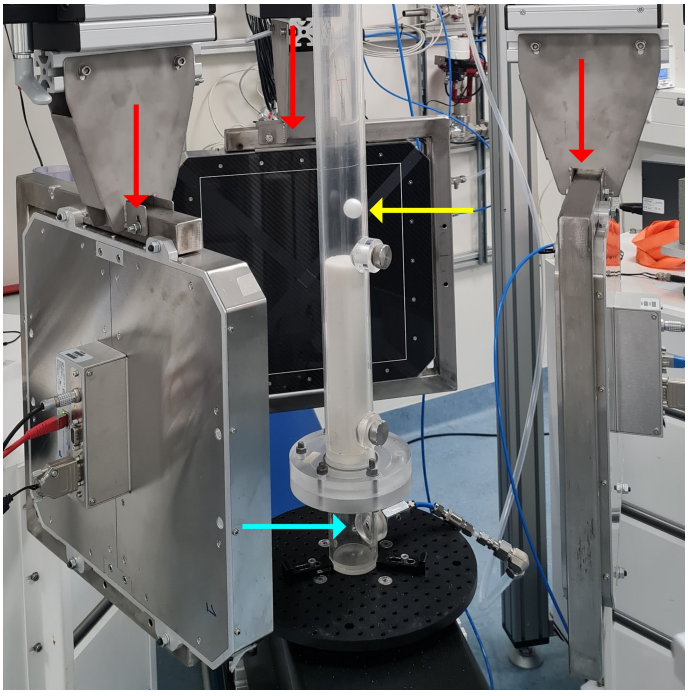


Figure 2.2: Photograph of the X-ray set-up with filled PMMA cylinder. Red arrows point to detectors, the yellow arrow to a polystyrene phantom (Section 2.5.2), and the blue arrow to the gas inlet. Sources are not in view.

X-ray source	Yxlon Y.TU 160-D06
X-ray source voltage	Max. 150 kVp
X-ray detector	Teledyne Dalsa Xineos 3131
Detector framerate	22 to 200 Hz
Detector resolution	1548×1524 pixels, 0.2 mm/pixel
Detector surface area	307×302 mm
X-ray source voltage	120 kVp
X-ray anode current	1.5 mA
Detector ROI	1548×550 pixels
Detector framerate	65 Hz
Object-source	935, 947, 937 mm
Object-detector	273, 268, 294 mm
Column material	Polymethyl methacrylate
Column diameter	50 mm inner, 60 mm outer
Particle material	Polystyrene
Particle size	560 µm (Geldart type B)
Minimum fluidization	15 cm/s

Table 2.1: Hardware specifications (top part) and configuration for the experiments in this work (bottom part).

rotation angles. For brevity, we denote all geometry parameters together with the tuple $\psi \in \mathbb{R}^{3\cdot 9}$. The object-source and object-detector distances, listed in Table 2.1, reflect the outcome of a careful manual measurement. They denote the distances between the column center and X-ray source, and column center and detector midpoint. To further increase the accuracy of ψ, we developed a workflow and methodology for a data-driven calibration procedure that allows for the simultaneous calibration of multiple sources and detectors. Compared to a manual measurement, this is both fast and flexible, and can accommodate both intentional and unexpected modifications to the set-up. In our experience, having an accessible calibration procedure significantly improves the quality of reconstructions.

The calibration is started by placing a marker object in the view of the three detectors. The marker object is sized similarly to a fluidized bed column, and contains a set of metal markers $\mathcal{I}$. The unknown coordinates of the markers are denoted by $\{m_i \in \mathbb{R}^3\}_{i\in\mathcal{I}}$. Using the rotation table, the marker object is then imaged from a small number of angles n_α. For three detectors, each marker $i \in \mathcal{I}$ is projected $3n_\alpha$ times, and thus yields a projection vector $p_i \in \mathbb{R}^{2\cdot 3n_\alpha}$. The vector is obtained by manually annotating the positions of the markers in the measurements.

To solve for ψ, a nonlinear least-squares optimization objective, Eq. (2.1), is set up. Here, the mapping $P_\psi \colon m_i \mapsto p_i$ encodes the geometric point-projection for multiple rotation-table positions. The Levenberg-Marquardt optimizer (see, e.g., [75]) is then used to solve the objective for all markers and ψ simultaneously:

$$\underset{\{m_i\}_{i\in\mathcal{I}},\psi}{\arg\min} \sum_{i\in\mathcal{I}} \|P_\psi(m_i) - p_i\|_2^2 \,. \tag{2.1}$$

Note that, for this procedure to work, it is not necessary to know the marker coordinates ahead of time. Since three or more rotation angles provide an abundance of projections, the marker positions $\{m_i \in \mathbb{R}^3\}_{i\in\mathcal{I}}$ may be optimized jointly with the sought parameters ψ. This conveniently allows the construction of marker objects for different column dimensions or detector regions of interest.

2.3.2.1 Data-driven calibration Our marker-based calibration (Section 2.3.2) allows the inference of geometry parameters ψ from the projection of markers. The parameters are encoded using the 3D positions of the sources and detectors, and the extrinsic Euler angles of the orientations of the detectors planes. Before our data acquisition, we scanned a fabricated PVC pipe with nine uniquely identifiable metal markers points. The pipe was positioned on a rotation table, and after imaging a full rotation, three angles were selected for annotation. We picked the three angles such that they are far apart, and such that the markers on the images do not overlap. This led to 9 projections for each marker, and a total of 81 marker projections.

The subsequent nonlinear optimization problem (see Eq. 2.1) was solved with the Levenberg-Marquardt [75] implementation in *SciPy*. The gradient of P_ψ was computed numerically. Since the distances between markers were not given *a-priori*, the found geometry parameters were determined up to a scaling factor. To see this, note that stretching the geometry (moving sources and detectors closer or farther) leads to the same projections. The known inner diameter of the PVC pipe was used to correct the scaling. For this, we first reconstructed the pipe in high resolution, using a full-angular scan from a rotation table. An ellipse was subsequently fitted in the horizontal reconstruction plane to yield the reconstructed diameter of the pipe. The comparison between the reconstructed diameter and the known diameter was used to determine the scaling factor.

2.4 Theory

2.4.1 Measurement principle

Data of fluidized beds is described by the measurement principle explained in the introduction, section 1.1. The aim of a tomographic algorithm is typically to infer μ, using Eq. (1.4) for many detector pixels and additionally formulated assumptions on $\mu(\eta)$. For a continuous medium, without sharp variations in its composition, μ can be assumed as a continuous function of space and time. However, in a fluidized bed, there are sharp variations at the interface between the particles and the gas, and, also there can exist local sharp variations along a ray, depending on the local distribution of the particles. Therefore, μ should be seen as a local-averaged quantity, averaged over a small volume and short exposure time, associated with the space and time resolution of the measurements

In our case, the imaging object consists of a PMMA cylinder, and a bubbling fluidized bed – a mixture of suspended solids and gas. The quantity of interest is not μ but the distribution of particles and gas. Similarly to μ, this can be expressed in terms of a local-

averaged gas fraction α, i.e., the local percentage of the volume that is occupied by the gas-phase (see, e.g., [76] and [77] for a formal definition). Note that α can vary between a minimum value, greater than zero, corresponding to the densest packing of particles, and a maximum value of one, corresponding to a pure gas. Alternatively, α can be normalized by defining a bubble fraction, b, with $b = 0$ corresponding to the average gas fraction in the *solids* (homogeneous dense phase) and $b = 1$ corresponding to the gas fraction in the *bubbles* (pure gas). I.e., b is defined as

$$b = \frac{\alpha - \bar{\alpha}_{\text{solids}}}{1 - \bar{\alpha}_{\text{solids}}}, \tag{2.2}$$

where $\bar{\alpha}_{\text{solids}}$ denotes the average gas fraction in the solids. Assuming that the attenuation in air is negligible, the relation between b and μ is given by

$$b = 1 - \frac{\mu}{\bar{\mu}_{\text{solids}}}, \tag{2.3}$$

where $\bar{\mu}_{\text{solids}}$ denotes the average attenuation in the solids.

Referencing method A referencing method is a technique to remove certain features, e.g., the PMMA cylinder, from a projection (Eq. (1.4)). When μ_{ref} denotes the attenuation of the unwanted features in the object, the desired projection should be from $\mu - \mu_{\text{ref}}$. Substitution in Eq. (1.4) gives

$$\begin{aligned} \int_l [\mu(\eta) - \mu_{\text{ref}}(\eta)]\,\mathrm{d}\eta &= \int_l \mu(\eta)\,\mathrm{d}\eta - \int_l \mu_{\text{ref}}(\eta)\,\mathrm{d}\eta \\ &= \log\left(I_0/I\right) - \log\left(I_0/I_{\text{ref}}\right) \\ &= \log\left(\frac{I_{\text{ref}}}{I}\right). \end{aligned} \tag{2.4}$$

This shows that the flatfield, I_0 in Eq. (1.4), can be replaced by a reference measurement I_{ref}, and that this removes the unwanted features. In Eq. (2.4), Beer-Lambert's law was applied twice, and I_0 dropped out of the equations under the assumption that I and I_{ref} were obtained in similar experimental conditions.

Projections of the bubble fraction Using the referencing method, we now derive a linear relation between b and projections of b, which we denote with y. A tomographic algorithm working with y therefore reconstructs the bubble fraction. Let I_{full} be a reference of the PMMA cylinder filled entirely with solids, i.e., $b = 0$ everywhere in the column, and

let I denote a measurement frame of an inhomogeneous bubbling fluidized bed. Then

$$\begin{aligned} y &= -\frac{1}{\bar{\mu}_{\text{solids}}} \log\left(\frac{I_{\text{full}}}{I}\right) \\ &= -\frac{1}{\bar{\mu}_{\text{solids}}} \int_l [\mu(\eta) - \mu_{\text{full}}(\eta)]\, \mathrm{d}\eta \\ &= -\frac{1}{\bar{\mu}_{\text{solids}}} \int_l [(1 - b(\eta))\, \bar{\mu}_{\text{solids}} - \bar{\mu}_{\text{solids}}]\, \mathrm{d}\eta \\ &= \int_l b(\eta)\, \mathrm{d}\eta. \end{aligned} \tag{2.5}$$

Since referencing removes the contributions of the PMMA cylinder, we have omitted it from the equation. Fig. 2.3 displays an example of I and I_{full}, and panel (a) of Fig. 2.9 shows the corresponding outcome, y. In the following, we explain how $\bar{\mu}_{\text{solids}}$ and I_{full} can be obtained experimentally.

Average attenuation in the solids Let ℓ_{int} be the intersection of a ray ℓ with the interior of a homogeneously filled PMMA cylinder. Then

$$\frac{1}{|l_{\text{int}}|} \int_{l_{\text{int}}} \mu_{\text{solids}}(\eta)\, \mathrm{d}\eta \tag{2.6}$$

describes the average attenuation along ℓ_{int}, and is, therefore, an estimate of $\bar{\mu}_{\text{solids}}$. In our set-up, we use the measurements of I_{full}, with a measurement of the empty PMMA cylinder, I_{empty}, acquired in the same experimental conditions (see Fig. 2.3). Applying the referencing procedure gives

$$\begin{aligned} \log\left(\frac{I_{\text{empty}}}{I_{\text{full}}}\right) &= \int_l [\mu_{\text{full}}(\eta) - \mu_{\text{empty}}(\eta)]\, \mathrm{d}\eta \\ &= \int_{l_{\text{int}}} \mu_{\text{solids}}(\eta)\, \mathrm{d}\eta \approx |l_{\text{int}}| \cdot \bar{\mu}_{\text{solids}}. \end{aligned} \tag{2.7}$$

By taking the ray through the center of the column, for which $|l_{\text{int}}|$ is known, we obtain an estimate of $\bar{\mu}_{\text{solids}}$.

Full column references for fluidized beds Eq. (2.5) requires a reference I_{full}, i.e., a cylinder with a bed that attenuated with $\bar{\mu}_{\text{solids}}$. Traditionally, the reference I_{full} is obtained with a packed bed, without any gas injection. However, this approach can introduce noise due to local sharp variations in the medium caused by the packing structure of the particles (see, e.g., [53]). An alternative is to use a homogeneously fluidized bed, where the motion of the particles reduces the noise, since the signal intensity is averaged over the exposure time. Moreover, using a fluidized bed leads to a more accurate determination of $\bar{\alpha}_{\text{solids}}$, because its value decreases due to the expansion of the bed, as shown in Fig. 2.4. The histogram of 0 cm/s, a packed bed, depends only on the mean attenuation and noise, whereas fluidized beds have a shifted distribution due to the expansion of bed.

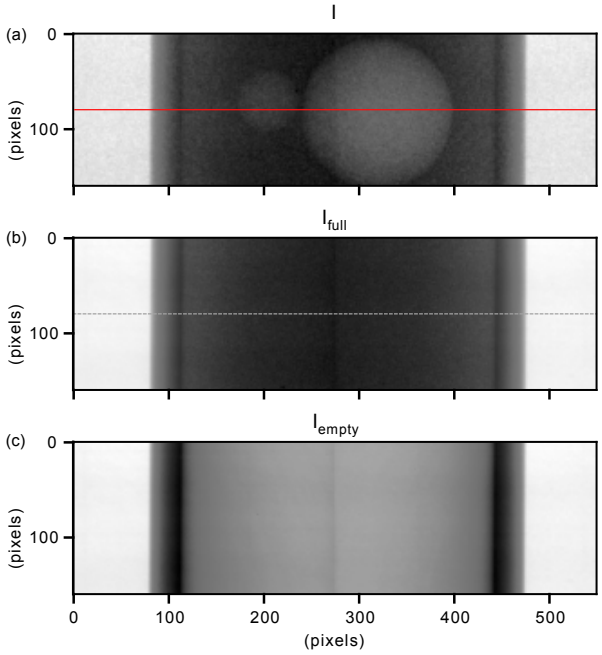


Figure 2.3: Raw measurements in a 550×160 pixel detector region, showing (a) two spherical phantoms (diameters equal to 10 mm and 23 mm), (b) a full column reference (not fluidized), and (c) an empty column. The red and dashed rows are used to display noise statistics in Fig. 2.11.

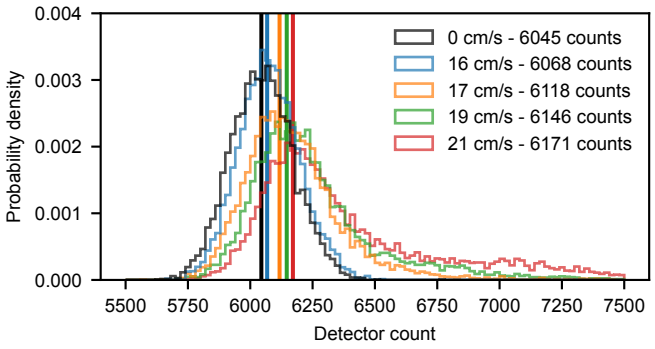


Figure 2.4: Detector count statistics for 1000-1200 measurements of a central pixel, for different superficial gas velocities of a fluidized bed. Vertical lines indicate the modes.

Measuring I_{full} in a fluidized bed is not straightforward, as the measurement during fluidization could contain bubbles. One possible strategy is to measure the column at a gas velocity right before the formation of bubbles. While this expands the bed somewhat, it can only be used for lower gas velocities, and can be challenging to achieve in practice. For our dynamic experiments in Section 2.5.3, we have therefore applied a heuristic approach, which uses the whole time series of projections I_t from an experiment to estimate I_{full}. The heuristic retrieves I_{full} by using a pixelwise mode of the data. To see this, note that a direct average or median value over all projection images does not yield an image of a full column, as bubbles increase the average X-ray intensity. Yet, when considering a single pixel of the detector, most values show a background (i.e., an X-ray that has passed through the bed without bubbles), and perturbations due to bubbles occur only in a relatively small number of frames. The pixelwise mode corresponds to the most commonly observed value, and therefore to the background I_{full}. It can be extracted by building a histogram of observed pixel values, and selecting the bin with the highest count, i.e., the *largest mode* of the distribution of I_t. We show this in Fig. 2.4, where the mode is computed using the maximum of an interpolating polynomial of degree 20 in the central detector pixel. Before reconstruction, we compute I_{full} for all pixels and detectors. Estimating I_{full} in this way is particularly useful when the bed expands significantly during fluidization, for instance with Geldart A type particles. Another advantage is that it does not involve acquisition of data using a secondary experiment, and therefore eliminates a possible error source. The disadvantage of such heuristic is that a reliable statistic requires a large number of time samples.

2.4.2 X-ray tomographic reconstruction

We refer the reader to Chapter 1 for the introduction on reconstruction techniques. Algebraic techniques are often used with sparse-angular or noisy data (see chapter 1 or [4]). SIRT suits our three-angle set-up, therefore, well. Analytic techniques, such as the filtered-backprojection (FBP), are faster, but cannot produce sufficient image quality in such an exceptional sparse-angular case. SIRT, in addition, serves as a baseline for more involved techniques with additional regularization and constraints.

The most straightforward way to obtain time-resolved reconstructions from fluidized beds is to run SIRT independently for each 3-tuple of projections, yielding a sequence of 3D reconstructions. Using the GPU-accelerated *ASTRA Toolbox*, reconstruction takes about a minute per timeframe. Alternatively, methods that reconstruct multiple frames simultaneously can incorporate spatio-temporal constraints, which can improve the image quality considerably. These constraints, however, often put more requirements on memory and computation time [78].

2.4.3 Visualization

In our results, we show measurements and reconstructions in a single format (see, for example, Fig. 2.5). In panel (a), we show a region of interest in the projections y, the result of Eq. (2.5). In panel (b), we display an iterate x of SIRT, Eq. (1.8), using a 3D density plot where the opacity is linearly increased from 0% to 20%. A horizontal and vertical cross-

sectional slice of x are shown in panel (c). Dashed lines are drawn between each source and the center of its corresponding detector.

2.5 Results

Tomographic reconstructions of fluidized beds provide, next to visual inspection, a better insight in bubble morphology and bubble dynamics than raw projection images do. While SIRT is commonly used for reconstruction, the unusually low number of projection angles in our set-up may cause distinct image artifacts [79]. These artifacts can usually be linked to parameter choices, the positions of bubbles, or the presence of noise. In Section 2.5.1, we investigate three commonly encountered artifacts through numerical simulations. In Section 2.5.2, we perform an experimental validation using spherical phantoms to mimic the bubbles in a fluidized bed. Section 2.5.3 presents experimental results of a Geldart B bubbling fluidized bed.

2.5.1 Numerical simulations

Numerical set-up In our simulations, we inspect reconstructions of solitary bubbles, of multiple bubbles, and of a bubble with noise. We use a simulation model that closely resembles our experimental set-up. In particular, we use the geometry of sources and detectors that is obtained after calibration of our set-up (Section 2.3.2). For each simulation, a ground truth is first constructed as a discretized volume. The projections, y, are then generated by the linear forward projection model for the conebeam geometry, i.e., the operator A in Eq. (1.5), via the implementation available in ASTRA Toolbox [80]. For reconstruction, we discretize on a $(300, 300, 1303)$ grid, with an isotropic voxel size of $550/300 \approx 1.83$ mm. SIRT is run with a masked area (cf., Eq. (1.8)) using a discretized exact cylinder of 50 mm diameter. In our experiments with real data, we use a larger mask to compensate for a slight tilt of the column. In all other aspects, the simulation model is equivalent to the reconstruction model in Section 2.5.3, and therefore enables experimentation with numerical data without the effects of noise, preprocessing, or calibration error. In the following, SIRT is run for 2000 iterations, and we do not apply any post-processing routines.

Solitary bubbles To show artifacts in the simplest circumstances, we start by studying *solitary* bubbles, i.e., bubbles for which a projection does not overlap with any of the other bubbles. In our experiments, this is frequently the case when no other bubbles travel at the same height. When modeled as spheres, solitary bubbles may be reconstructed to surprisingly high accuracy. Fig. 2.5 shows that three projections of a full-gas spherical bubble are sufficient for an accurate reconstruction; only minor perturbations are visible at the bubble interface in the directions of projection. Of notable interest is the horizontal cross-sectional plot, where, despite the three-angle view, the bubble boundary is almost perfectly round.

We found that only in the case of noiseless data and $b = 1$, SIRT recovers the bubble to this level of accuracy. In many other situations, bubbles are reconstructed as irregular convex hexagons, or have a cupping-like distortion of the bubble fraction. An example is given in

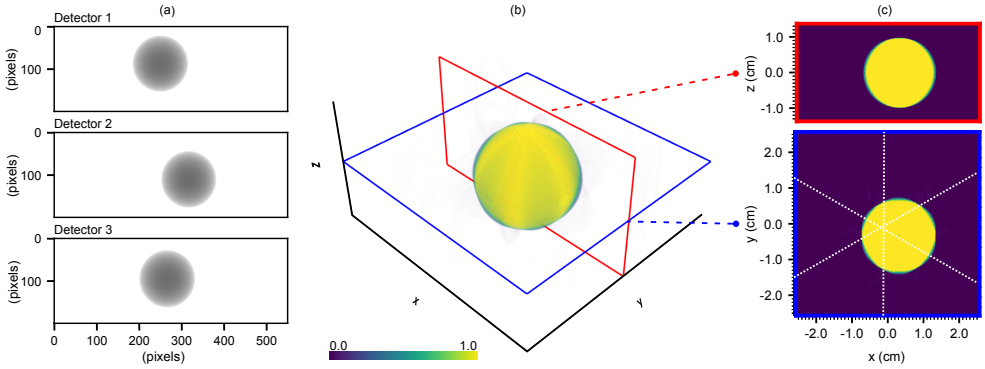


Figure 2.5: Reconstruction of a simulated sphere with a diameter $D = 20$ mm and a bubble fraction $b = 1.0$. The visualization format is explained in Section 2.4.3.

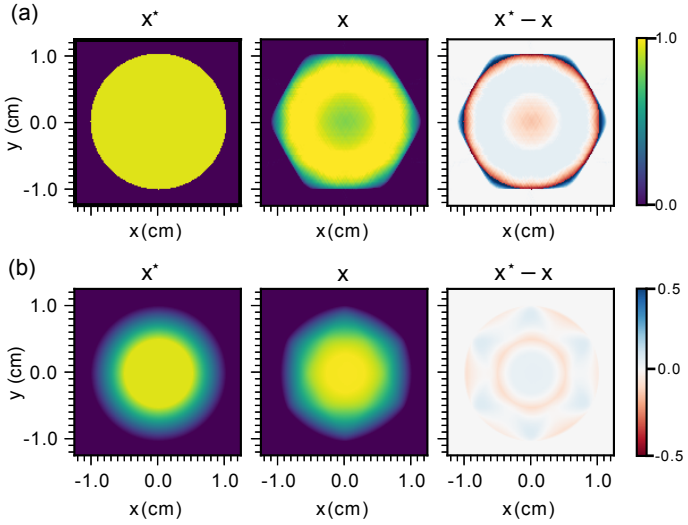


Figure 2.6: Reconstruction x of a spherical bubble $x^\star$ with $b = 0.95$ and diameter $D = 20$ mm. The bubble in panel (b) has a smoothed interface and is less affected by reconstruction artifacts.

panel (a) of Fig. 2.6, where $b = .95$ rather than $b = 1$. Fortunately, this does not always pose a problem. The artifact only shows in the horizontal plane, where limited information is present due to the sparse-angular resolution. The severity of this artifact furthermore depends on the bubble fraction, the noise on the projections, and the effect of the SIRT constraints, i.e., $0 \leq x \leq 1$ in Eq. (1.8). In particular, artifacts are less pronounced when there is some spatial variation in the bubble fraction. A bubble with a smoothed interface, shown in panel (b) of Fig. 2.6, is less affected by hexagon and cupping artifacts. This shows that the use of additional regularization and constraints has the potential to significantly improve the quality of the results.

Bubble-bubble reconstruction interference Reconstructions become more complicated when projections contain overlapping bubbles. A reconstruction of multiple bubbles is different from the addition of reconstructions of solitary bubbles: while the X-ray model is linear (Eq. (1.5)), the constraints in Eq. (1.7) lead to a non-linear relationship between y and $x^\star$. Fig. 2.7 illustrates this with a simulation in which, on Detector 3, a smaller bubble overlaps with a larger bubble. Intuitively speaking, SIRT relies on Detector 1 and Detector 2 to separate the bubbles spatially, however, cannot do that uniquely. In the figure, intensities from the sides of the bigger bubble are therefore erroneously displaced into small patches, and both bubbles show hexagon artifacts. The problem is more pronounced with multiple overlapping bubbles, but the magnitude of the artifacts depend on the shapes, distances, and bubble fractions.

Geometry of noise artifacts The last simulation clarifies the origin of a star-shaped streaking artifact, a phenomenon that was previously observed for our double line detector set-up [53]. We confirm that one cause of this artifact is noise, through a simulation of additive Gaussian noise on the projections. In experimental data, other error sources, i.e., mismatches between Ax and y in Eq. (1.5), could also lead to similar artifacts. Fig. 2.8 shows the reconstruction of a spherical bubble, with a diameter $D = 10$ mm, a bubble fraction $b = 1.0$, in which a zero-mean Gaussian noise with variance equal to $2.6 \cdot 10^{-2}$ was added to the projections. Panel (a) shows the average reconstruction, panel (b) the bias, and panel (c) the variance, using 100 reconstruction samples. The figure shows increased variance along projection lines and around the bubble, which is again an effect of non-linear reconstruction: in a linear method, the spatial distribution of the variance would not depend on x, and hence a variance plot would not show a bubble. This does, however, not suggest using one: a linear method would have higher variance. At the end of the following section, we quantify the variance of the noise in our set-up.

2.5.2 Phantom experiments

We proceed with an experimental validation of our set-up. First we repeat the bubble-bubble reconstruction of last section, using two spherical phantoms. We then validate the accuracy of velocity estimations, and compute noise statistics from our measurements.

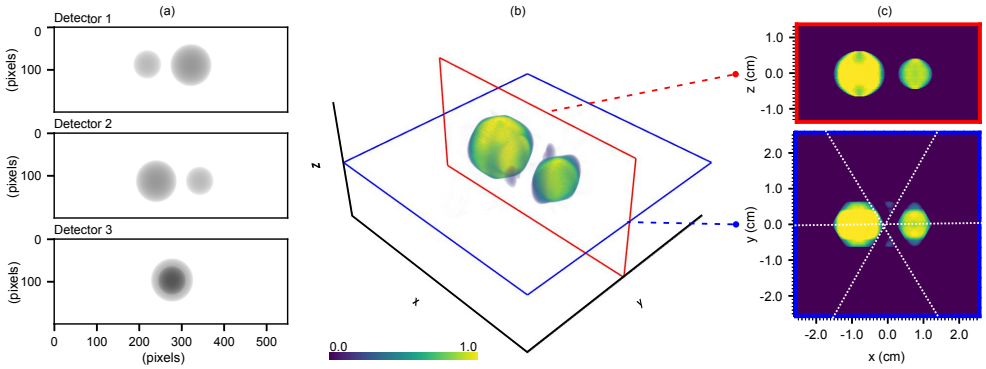


Figure 2.7: Reconstruction of two simulated bubbles with diameters $D_1 \approx 1.23$ cm and $D_2 \approx 0.83$ cm (74 and 50 voxels, respectively), for which the projections overlap on Detector 3. The visualization format is explained in Section 2.4.3.

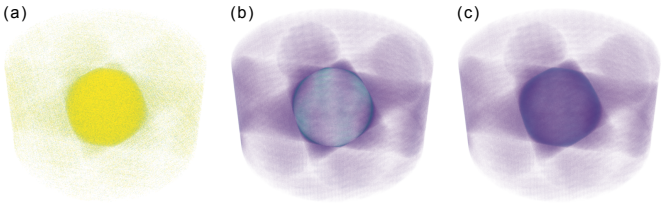


Figure 2.8: Reconstruction (a), bias (b) and variance (c), from a simulated spherical bubble, with a diameter $D = 10$ mm and a bubble fraction $b = 1.0$. Zero-mean Gaussian noise with a variance of $\sigma^2 = 2.6 \cdot 10^{-2}$ has been added to the projections. (b) and (c) are computed using 100 samples.

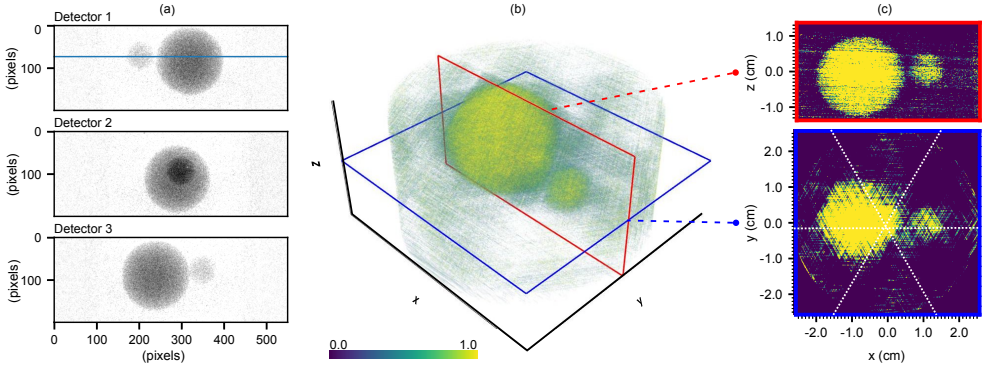


Figure 2.9: Reconstruction of two polystyrene phantoms, with diameters $D_1 = 23$ mm and $D_2 = 10$ mm, that overlap on the second detector. The reconstruction, panel (b), has been rotated to match Fig. 2.7. The detector row marked in blue is used to compute noise statistics in Fig. 2.11. The visualization format is explained in Section 2.4.3.

Static bubble phantoms Fig. 2.9 displays a two-ball phantom experiment that repeats the numerical simulation on bubble artifacts (Fig. 2.7). Here, two expanded-polystyrene foam balls are imaged in a 50 mm diameter column, filled with Geldart B particles (specifications in Table 2.1), without the inflow of gas. An example of a phantom is displayed in Fig. 2.2. The phantoms are spherical, feature sharp boundaries, and have negligible X-ray attenuation. They therefore model idealized bubbles in a static snapshot of a fluidized bed. The column is rotated, using a rotation table, so that the smaller, 10 mm phantom, overlaps with the larger, 23 mm phantom on Detector 2. Without fluidization, the static packing structure of the solids adds an additional noise-like pattern to I and I_{full}. We will further discuss this in Section 2.5.3.

In this experiment, Fig. 2.9, the larger bubble in the horizontal cross-sectional plot of x displays the hexagon artifact, as well as a slight decrease in intensity. The spherical shape is still recovered well, thanks to the high resolution of the detectors in the vertical dimension. In the 3D plot, a few low-intensity patches are observed between the two bubbles, although not as pronounced as in the numerical example. The cross-section cuts in Fig. 2.9 show the appearance of streaks, which are associated with the noise. In essence, comparing figures 2.9 and 2.7, we see that the reconstruction of the phantom-bubble is a more fuzzy version of the numerical bubble, and Fig. 2.9 can be interpreted as a representative signature of an actual (spherical) bubble.

Bubble diameter estimation To explore our set-up's ability to reconstruct spheres, we attempt to recover the known diameters of the 23 mm and 10 mm phantoms, as well as two simulated counterparts, using a template-matching procedure. Letting $B \in \mathbb{N}^{N_d^3}$ denote a binary template of a 3D sphere with diameter d, we compute a matching score between the template and a reconstruction volume x with

$$\text{score} = \frac{1}{N_d^3} \sum_{i=1}^{N_d^3} \Big(x_i B_i + (1 - x_i)(1 - B_i) \Big). \tag{2.8}$$

The first term in the summation matches the sphere, consisting of 1.0-valued voxels, whereas the second term matches the background, with 0.0 values – this avoids matching a template of a small sphere with a large bubble. Eq. (2.8) simplifies to the Dice similarity coefficient when x is binarized, and can take values between 0.0 and 1.0. The score is 1.0 in the case of a perfect match with the template. As $0 \leq x \leq 1$, we can omit binarization, which avoids the subjective choice of a segmentation threshold parameter, and lets gray values contribute to the score with a value that is proportional to their distance to the template.

Fig. 2.13 shows template-matching results for both the phantom and simulation experiment. A spherical template with diameters ranging between 1 and 35 mm is translated through the volume. The best score encountered during this translation is plotted in panel (a) as a function of diameter. In the noiseless simulation, bubble diameters are recovered to a high accuracy, which suggests that the impact of hexagon artifacts is limited in this experiment. In the phantom experiment, the 10 mm phantom is underestimated as 9.1 mm

and the 23 mm phantom is overestimated as 23.5 mm. Their respective scores of 0.70 and 0.86 reflect a substantial impact of noise. Panel (b) shows a volume rendering of the best matching templates together with the reconstructed images of the phantoms, and panel (c) shows an intersection in the horizontal plane with the simulations in the lower half. Note that in panel (c) diameters are plotted smaller than found, as the intersection plane lies in-between the two phantom centers.

The experiment is an indication that the technique has the potential to quantify bubble volumes within a reasonable error margin, likely due to the information that is present in the bubbles' 3D neighborhoods. To explore how reliably the shapes of real bubbles can be reconstructed, phantom experiments with more complex shapes (e.g., concave or ellipsoid forms) will be designed in a future study.

Moving phantom Fluidized beds are reconstructed by computing x_t for a 3-tuple of projections at each time t. However, to extract dynamic characteristics of bubbles, such as the bubble velocity, measurements need to be taken at prescribed (equidistant) intervals and with synchronized detectors. This aspect of the set-up is tested end-to-end, by performing a time-resolved reconstruction of a moving phantom. First, a 25 mm glass phantom is pulled upwards through an empty column, using a motor-driven traverse. Then, in each timeframe t, a position $p_t \in \mathbb{R}^3$ of the phantom is inferred by fitting a 3-dimensional sphere to the reconstruction x_t. The experiment was repeated for traverse speeds of 62 and 125 mm/s and for three different horizontal starting positions of the phantom.

Fig. 2.10 shows the 125 mm/s experiment, with the phantom starting in the center of the column. The plot confirms an accurate upward speed of the phantom. Using a least-squares interpolation of p_t, the phantom was estimated to have a speed of 128 mm/s, an overestimation of 2.4%. This error was consistent for all other starting positions of the phantom. The results of the 62 mm/s experiment were similar, now with a overestimation of 1.6% in all starting positions. Since the error correlates with the traverse speed, but not with the phantom starting position or travel distance, we expect the error to be due to an inaccuracy in the experimental equipment.

Measurement noise The local sharp variations in the medium, associated with the distribution of the particles, leads to high noise. In a fluidized bed, due to the motion of the particles, the averaging of the signal over the exposure time reduces the noise. However, to capture the fast dynamics of fluidization, detectors need to operate at short exposure times. As a consequence, the projection data still contains high levels of measurement noise, and our reconstructions degrade accordingly (as shown in Fig. 2.8). Measurement noise can be attributed to stochastic fluctuations due to photon scattering and absorption, as well Poisson-distributed detector noise [3]. To quantify how well bubbles can be observed, we compute noise statistics over 1,000 measured frames of the two static bubble phantoms. We take the pixels in a single detector line, displayed in red in panel (a) of Fig. 2.3, and in blue in panel (a) of Fig. 2.9.

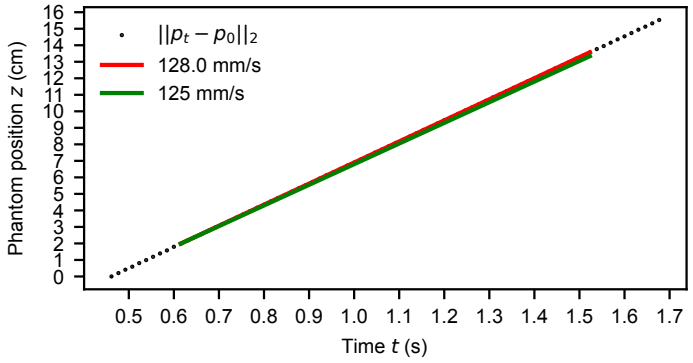


Figure 2.10: Least-square interpolation of positions p_t, obtained after presetting the traverse to 125 mm/s, and the detectors to 65 Hz.

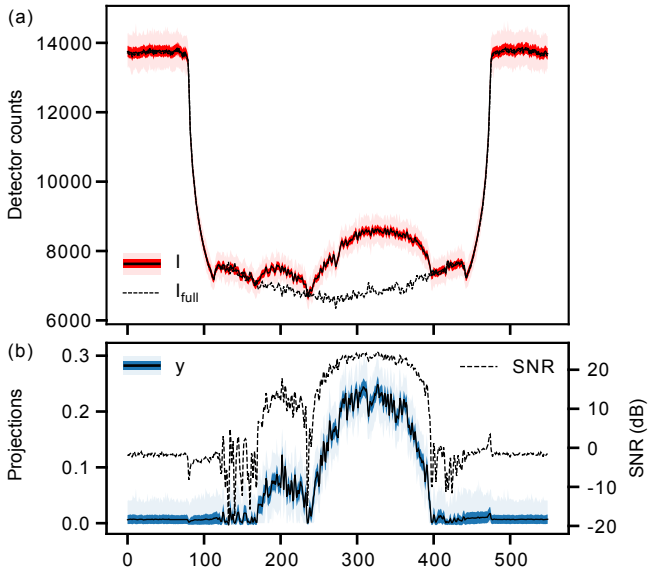


Figure 2.11: Panel (a) shows the line profile of raw measurement data of the marked detector row in panel (a) and (b) of Fig. 2.3. The computed noise statistics are shown in red (std.dev. in strong red, min/max values in light red). Panel (b) shows the same detector row, now after preprocessing, see the marked row in panel (a) of Fig. 2.9. Noise statistics of y are in blue.

Fig. 2.11 plots I_{full}, I, and y, and their means and standard deviations. For I, i.e., the raw data, detector counts range from 6,500 to 9,000 in the solids. The corresponding standard deviations vary from about 130 counts, near the inner wall of the PMMA cylinder, to about 110 in the column center. The standard deviation increases in bubble voids, and depends on the size of the void. This is expected, as the variance of counting noise depends on its mean value, which is proportional to the photon flux, and thus inversely proportional to attenuation.

Panel (b) of Fig. 2.11 shows the impact of the random fluctuations on I and I_{full} after computing y from Eq. (2.5). The signal-to-noise ratio for the signal y_i in a pixel i is computed as

$$\text{SNR}_i = 10 \log_{10} \left(\frac{\mu_i^2}{\sigma_i^2} \right), \tag{2.9}$$

where μ_i is the mean and σ_i the standard deviation. This yields an SNR value of 24 dB for the 23 mm phantom, and an SNR of 12 dB for the 10 mm phantom, both measured at the center of the phantom. This confirms the intuition that we should be able to image larger bubbles with a higher confidence. It must be noted that the SNR values are a set-up dependent indication of image quality. Noise varies with the detector framerate, source current, column diameter, and set-up geometry. Furthermore, since bubbles are observed from multiple angles, we expect better signal-to-noise ratios in reconstructed images. An explicit variance analysis of noisy reconstructions, e.g., as conducted for a parallel-beam geometry using the filtered-backprojection [3], cannot easily be applied to our custom geometry and particular image reconstruction method. Unlike the circular parallel beam case, with three source-detector pairs, the voxelwise variance in our reconstructed volume may not be radially symmetric. Moreover, with box constraints in the iterative method, noise will be nonlinearly propagated over the 3D reconstruction, leading to unclear data-dependent transformations to the per-voxel noise distributions. The reader is referred to Chapter 9.8 of Buzug [3].

In our multi-source set-up, one contribution of noise is expected due to cross-scattering of photons, i.e., photons that originate from the non-facing sources on the left and right sides of each detector, and have scattered under an angle of approximately 120°, primarily via Compton scattering (for a 150 kVp X-ray spectrum). To quantify this error, we measure detector counts with the facing source off, and with one or two of the non-facing sources on, and subsequently average over all pixels in 630 frames and all symmetrical source-detector combinations. The detector counts, listed as S and D in Table 2.2, are the contributions due to scattering and darkfield current (Section 2.3.1). Their combined value can reach up to 14% of the measured signal (i.e., $I \approx 6500$, see Fig. 2.11). The scattering of the particulate material has a relatively small contribution compared to the PMMA column, and the dynamic variation of photon scattering during fluidization is therefore expected to be small. In a reconstruction computed with scattering and darkfield correction, we noted only subtle changes in voxel values, between -0.01 and 0.01. This did not lead to a structural improvement in the background, bubbles, or artifact shapes. Removing a uniform identical bias from both I and I_{full} in Eq. (1.4), with $I_{\text{full}} < I$, leads only to a slightly stronger

	One source			Two sources		
	D	*S*	*Rel.*	*D*	*S*	*Rel.*
No column	324	135	7%	324	266	9%
Empty column	324	252	9%	324	501	13%
Full column (not fluidized)	324	302	10%	324	603	14%

Table 2.2: Photon scattering originating from the X-ray sources not facing a detector, e.g., photons originating from source 2 and 3 that are scattered into detector 1, cf. Fig 2.1. Listed are detector counts measured when either a single or both of the non-facing sources are switched on. The values S (scattering) and D (dark current) are averages over all detector pixels and all possible source-detector combinations. *Rel.* denotes the percentage of $D + S$ with respect to the minimum signal, taken as $I := 6500$ detector counts.

contrast between the intensities. Hence, the result suggests that the stochastic fluctuation of noise has a larger impact on the quality of reconstruction than the scattering bias. A future study to mitigate the impact of scattering, using advanced noise correction techniques [58], is warranted.

2.5.3 Geldart B bubbling fluidized beds

In the following experiment, the Geldart B bed is brought to the fluidization regime using a superficial gas velocity of 17 cm/s. This combination of particle size and gas velocity leads to distinctive bubbles, and is therefore suitable as a demonstration of our technique. At lower superficial gas velocities, bubbles are smaller, whereas at higher velocities, the flow transitions into the slugging regime. X-ray projections are recorded above the gas inlet (see Fig. 2.14 for a single frame). SIRT was run with 200 iterations as stopping criterion to prevent fitting to noise. Detector framerates were set to 65 Hz, i.e., an exposure time of 15 ms. This is the maximum framerate, which depends on the detector ROI. It can be increased by changing the source-detector distances or by imaging with a smaller column – although the latter would increase bubble interactions with the column wall. The supplementary video (Fig. 2.16) in the publication associated with this chapter animates frames 450 to 650, corresponding to three seconds of the experiment. The interval contains differently shaped and sized bubbles, as well as a variety of dynamical behaviours.

Fig. 2.12 shows a solitary bubble extracted from timeframe 475. Together with Fig. 2.9, this confirms that (i) the bubbles are reliably localized and reconstructed, and that (ii) the artifacts (streaks and hexagon-shapes) are an intrinsic limitation of the sparse-angular reconstruction. In comparison to phantom experiments, hexagon artifacts are less pronounced in fluidized beds, due to a smoother variation of b at the bubble interface (cf. Fig. 2.6). (Smoothly varying and irregular interfaces are known physical phenomena [81].) A second observation is that projections in Fig. 2.12 panel (a) contain less noise than in Fig. 2.9 panel (a). This is a combined effect of the creation of a full column reference (Section 2.4.1) and an averaging effect due to the motion of the solids during the exposure time of a single detector timeframe.

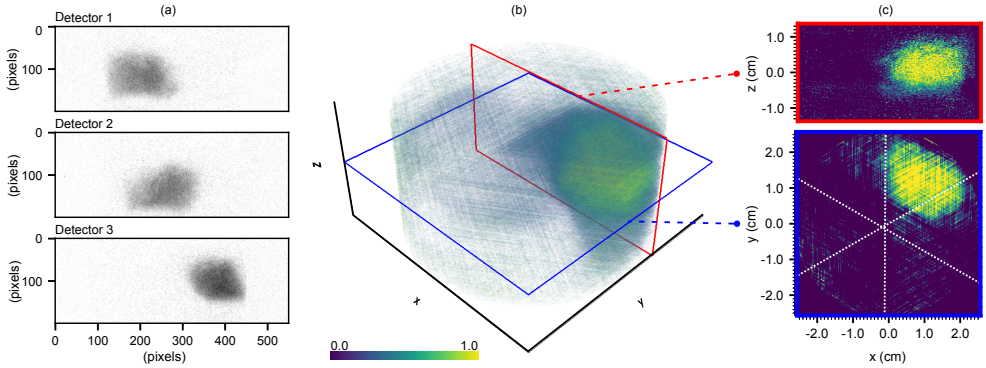


Figure 2.12: A solitary bubble that is extracted from frame 475 of the Geldart B fluidized bed reconstruction with a superficial gas velocity of 17 cm/s. The visualization format is explained in Section 2.4.3.

The video, and Fig. 2.15 with timeframes 482-488, display the dynamic behaviour of the field $b(x, y, z, t)$. This enables the quantification of bubble dynamics [81]. We observe that:

- Small bubbles form at the gas inlet and merge with other bubbles on their paths upwards through the column. The cores of the larger bubbles are usually reconstructed with full gas ($b = 1$, yellow), but, as before, bubble interfaces are not sharp (blue-green).

- The flux of interstitial gas through the solids plays an important role, leading to: (i) the existence of regions where b takes "intermediate values"; (ii) the growth and/or shrinkage of bubbles; (iii) the coalescence and/or breakup of bubbles and; (iv) the non-local interactions between bubbles.

- While bubbles often appear to be convected in quasi-steady ways upward, the dynamics are in fact 3-dimensional and intrinsically unsteady, as they are driven by the flow of gas through the column. Two examples are visible in Fig. 2.15. In the middle, a large bubble emerges, while in the top a smaller bubble disappears, i.e., disperses in the solids. Both occur in a very short time, and with little upwards displacement.

Overall, the Geldart B fluidized bed experiment shows the potential of our technique for imaging complex 3D dynamics. A pseudo-3D technique, i.e., using a vertical stacking of 2D slices over an interval of time [82, 83], would not have been adequate because of the unsteady 3D dynamics. To see this, note that in the pseudo-3D technique, subsequent 2D reconstructions of the bubble (while it travels through the slice) are stacked vertically. This only enables an accurate representation of the bubble when the shape of the bubble can be assumed constant during its path upward.

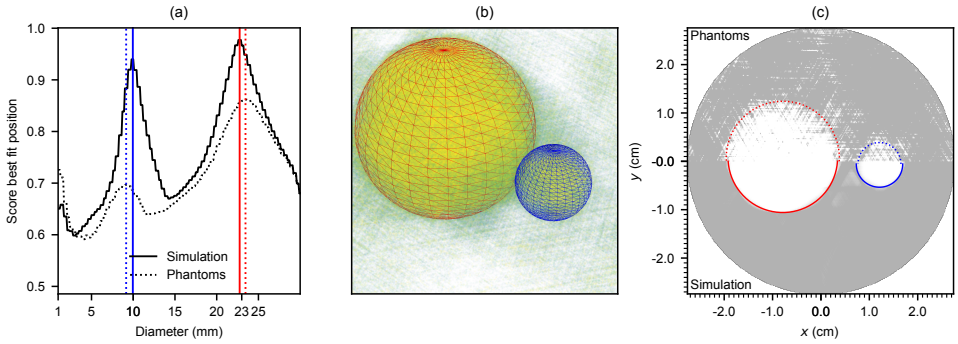


Figure 2.13: Template-matching of spheres with varying location and diameters to 10 mm (blue) and 23 mm (red) diameter bubbles, in a phantom experiment (dotted line) and a noiseless simulation (solid line). Panel (a) shows the best matching-score, Eq. (2.8), obtained for each diameter, panel (b) a 3D plot of best-matching templates overlayed onto the reconstruction of the phantoms (see Fig. 2.9), and panel (c) a horizontal cross-section of the reconstructions of noiseless simulations (bottom part) and phantoms (upper part) in grayscale coloring, onto which the boundaries of the best-matching spheres have been drawn.

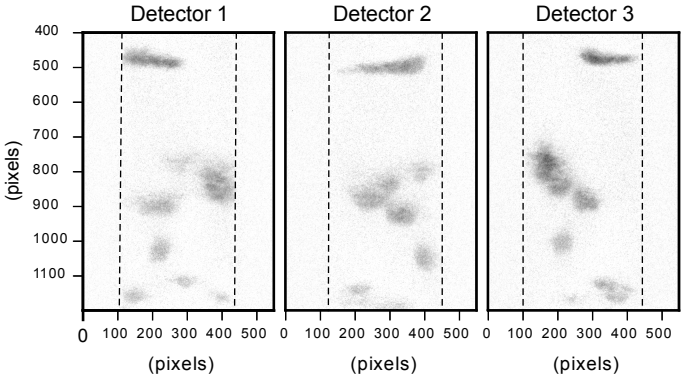


Figure 2.14: Projections y of timeframe 485 of the bed fluidized with a superficial gas velocity of 17 cm/s. The projections have been cropped to a region of interest, showing rows 400-1200. The column walls are indicated with dashed lines.

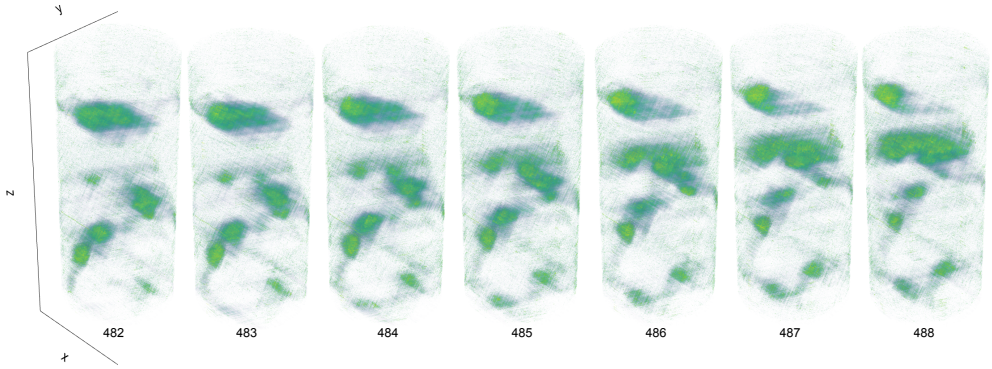


Figure 2.15: Timeframes 482-488 of the bubbling fluidized bed using an superficial gas velocity of 17 cm/s. For this visualization we enhanced the images by postprocessing with a low-weight ($\lambda = 1/0.065$) total variation filter.

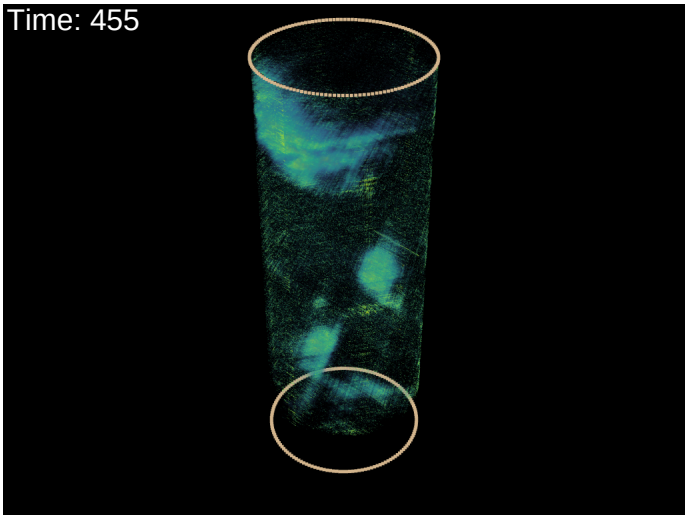


Figure 2.16: Still of the 17 cm/s experiment. In the movie, we enhanced the reconstruction by postprocessing with a low-weight ($\lambda = 1/0.065$) total variation filter.

2.6 Conclusion

To measure the gas-solids mixture of fluidized beds, various techniques are used in the literature, each with its own advantages and limitations. In this article, we have introduced a new technique that is fully-3D and simultaneously achieves a high temporal resolution. It consists of a set-up with three stationary X-ray source and flat panel detector pairs, a data-driven geometry calibration procedure, and a tailored image reconstruction method. The technique enables dynamic imaging of large 3D regions, is conceptually simple, and can easily be modified – for instance to image at a high framerate in a region of interest. It therefore enables straightforward experimentation with varying column sizes and different bed materials. Future considerations are an upgrade of the hardware, e.g., an expansion to five source-detector pairs (although this would be a costly and mechanically complex task) and the implementation of an additional beam hardening correction for imaging larger diameter columns [84].

In comparison to an X-ray source and detector on a gantry, such as is common in medical CT, our set-up has a high framerate, but requires a sparse-angle reconstruction for each timeframe. To study this aspect, we have used SIRT as a baseline in numerical simulations and phantom experiments, and analyzed the noise-induced artifacts, hexagon-shaped bubbles, and the degradation of resolving power when multiple bubbles occur at the same detector height. The artifacts are 3D analogues of observations in our previous line-detector set-up [53, 85], and provide directions for tailored post-processing approaches and more sophisticated reconstruction algorithms. Since noise was found to have a substantial effect in the reconstruction, we expect noise-suppressing priors to improve the image quality considerably. We will see that, in Chapter 5, indeed denoising as a preprocess improves the SIRT reconstructions. Depending on the bubbling regime, priors could also incorporate information on the sparsity of the bubbles, their solid contents, or the nearby timeframes. This has the potential to reduce artifacts and achieve a more precise determination of bubble interfaces.

In our results of a Geldart B bubbling fluidized bed, we find clear correspondences between the time-resolved experiment and conducted phantom experiments. Compared to our previous line-detector set-up, we found that the set-up is able to capture the complex 3D evolution of bubbles in the entire volume. This shows the potential of the technique for better quantitative analysis, such as average gas hold-up, as well as a better qualitative analysis, such as coalescence- and break-up processes, of bubbling fluidized beds.

Chapter 3

Just-in-time Deep Learning For Real-time X-ray CT

This chapter is based on the article: Adriaan Graas, Sophia Bethany Coban, K. Joost Batenburg, and Felix Lucka. "Just-in-time deep learning for real-time X-ray Computed Tomography". In: *Scientific Reports* 13.1 (2023), p. 20070. ISSN: 2045-2322. DOI: 10.1038/s41598-023-46028-9

3.1 Introduction

Computed Tomography (CT) is a powerful imaging technique to reveal the interior of objects using X-rays, with applications in health care, industry, life sciences, physics [87], material sciences [88], as well as many other fields [89]. At synchrotron light sources and X-ray microscopy laboratories, time-resolved tomography allows the reconstruction of dynamically evolving processes [90]. In these environments, experimental data is collected by an imaging scientist, but reconstruction is often postponed to a later stage. As a result, the domain expert has little control or feedback over the imaging process [9, 12], and an experiment may need to be repeated after reconstructions have been inspected. With recent advances in hardware, such as GPUs (*Graphical Processing Units*) and CMOS (*Complementary Metal Oxide Semiconductor*) detector technology, a small number of synchrotrons and laboratories have now developed *real-time* tomographic pipelines. Next to streamlining data acquisition and preprocessing steps, these pipelines achieve reconstructions within milliseconds [9, 91, 92, 93]. Live observation of the experiment helps image scientists and domain experts to optimize acquisition settings, and therefore save valuable time and storage [90, 92, 94].

While real-time reconstruction provides a valuable insight into the imaged object through the spatial distribution of X-ray attenuation, *image processing and analysis* tasks are often necessary to improve or evaluate the experimental outcome. Algorithms for these tasks can perform image enhancements, such as noise and artefact removal, and show whether or not the reconstructed object features are of sufficient quality. In more complex cases, they may extract semantic information, e.g., through object classification, counting of object instances, or image segmentation. This helps the domain expert in the evaluation of the experiment. However, many image analysis tasks take too much time when computed with traditional algorithms, and integration in tomographic pipelines has therefore not yet been possible. With the advent of Deep Learning, this has changed. A new class of algorithms called *Deep Neural Networks* (DNNs), in particular those using convolutional layers (CNNs), shows remarkable results for a wide range of image tasks, such as motion estimation and image classification [35]. Because DNNs can operate on the timescales required by real-time tomography, they are a promising next component in the tomographic pipeline, which we illustrate this in Fig. 3.1. In the nearby future, DNNs may be able to close the experimentation loop, by providing automated feedback and experimental control [92]. For example, a DNN could automatically identify a region of interest, and adapt the scanning geometry to zoom into it.

In practice, applying DNNs in real-time tomographic pipelines is not straightforward. The success of DNNs is generally attributed to *training*, i.e., the optimization of algorithmic hyperparameters based on data. This process takes place before *inference*, i.e., the application of a network to unseen data. The real-time setting makes training challenging for two main reasons. The first concerns the acquisition of training data. Image reconstructions – the inputs to a neural network – depend on the object to be imaged, the set-up and noise levels, and the reconstruction algorithm. Experiments are, furthermore, often not repeatable, and the effect of user interaction with the experiment cannot easily be anticipated as

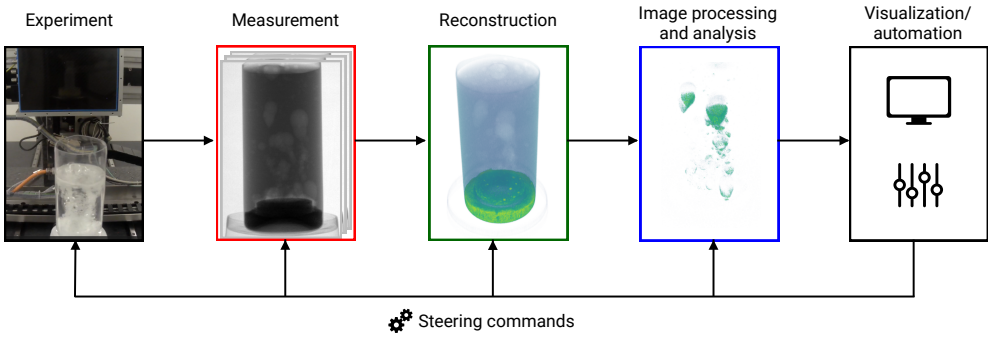


Figure 3.1: In real-time tomographic pipelines, measurement acquisition, image reconstruction, and image processing or analysis, are subsequent computational steps that can be executed concurrently during an experiment. User feedback or DNN automation can be used to steer the experiment. The illustration shows a dissolving-tablet experiment at the FleX-ray laboratory [9], and is further detailed in the Experimental set-up section.

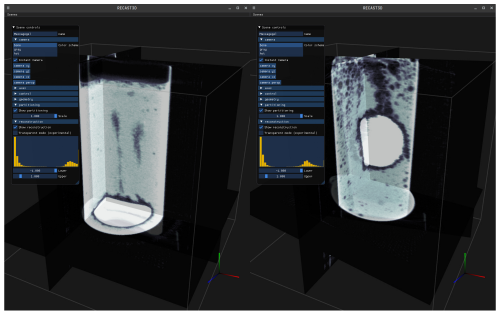


Figure 3.2: Illustration of the RECAST3D user interface (`https://github.com/cicwi/RECAST3D`, release v1.1.0). Left: *Experiment (A)* with fast dynamics. Right: *Experiment (B)* with slow dynamics. During the experiments, reconstructions are performed and visualized on three slices.

well. Generating training data of sufficient quality and quantity from previous experiments may therefore be difficult, or relevant training data may simply not be available. The second reason is a lack of training time. Beamline time at synchrotron facilities is scarce and expensive, making schedules too tight to train a DNN in between experiments.

We propose to overcome these challenges by taking advantage of the spatio-temporal continuity of samples from a tomographic pipeline. Unlike typical data sets used to train CNNs, image samples from the pipeline are the result of continuous processes, e.g., the dynamics of the experiment, the mechanics of the set-up, or changes in the reconstruction parameters. Data samples are therefore not i.i.d. samples, but rather possess a high degree of spatio-temporal continuity. This opens the opportunity to train small neural networks that adapt to the data during the experiment, while simultaneously being used for inference on incoming reconstructions. Since patterns in the images are likely to reappear in the following seconds to minutes, we say the network is trained *just-in-time* for inference. This offers the sought flexibility towards changing and unpredictable reconstructions, and can be fully automated, once integrated in a pipeline. The approach is limited to network architectures that can be trained quickly, and is only applicable to learning tasks that do not require external data. However, as the goal is to perform well on the current image from the pipeline, only a small data set is required for training. To the best of our knowledge, we are the first to propose and investigate such an approach for CNNs.

In this article, we explore the new concept, with the aim to enable real-time tomography with Deep Learning-based image processing in tomographic pipelines. For this purpose, we replay two experiments with the RECAST3D software, using data obtained from our FleX-ray laboratory at CWI, in Amsterdam [9]. The experiments entail tablets that are dissolved in a fluid inside a glass container with a granular ground (see Fig. 3.1, 3.2). *Experiment (A)* features large spatial structures and fast dynamics, whereas *Experiment (B)* has smaller structures and slower dynamics. They are representative for the study to bubble physics, an important topic in material and engineering sciences [95], as well as dissolution processes, which is relevant in pharmaceutical research [96]. Due to natural noise and artefacts, the experiments allow for a real-world challenge of our learning strategy. The article is organized as follows: First, in Methods, we formalize real-time tomography, and explain the Deep Learning context and self-supervised denoising task called Noise2Inverse [40]. We furthermore discuss our software set-up and software contributions. In the subsequent section on Just-in-time Learning, we introduce three topics that we identified as main challenges: the stochastic structure of real-time data, suitable network architectures, and an online learning strategy. In the penultimate section, Results, we report on the findings for each topic with DNNs trained on our experimental data. We finalize with a discussion of the potential and remaining challenges of just-in-time learning.

3.2 Methods

3.2.1 Real-time Computed Tomography

Tomographic reconstruction is an imaging technique that probes the interior of an object using a penetrating beam, such as X-rays or electrons. In real-time X-ray CT, radiographic projections $\mathbf{f} \in \mathbb{R}^{m_x \times m_y}$ are recorded in a continuous stream while the object is rotated. A reconstruction algorithm then takes the last m_ϕ projections from the stream, usually corresponding to a full rotation, and recovers the interior of the object as a three-dimensional image $\mathbf{x}_t \in \mathbb{R}^{n_x \times n_y \times n_z}$. The reconstruction algorithm achieves this by inverting the X-ray transform F. That is, it solves $\mathbf{x}_t$ from the relation $[\mathbf{f}_{t-m_\phi}, \ldots, \mathbf{f}_t] = F(\mathbf{x}_t)$, with t the index of the last projection. Two major classes of reconstruction algorithms are used in practice. *Direct* algorithms solve the inverse problem analytically, by formulating the solution in a closed-form equation. Iterative algorithms, on the other hand, express the inverse through an optimization objective. Commonly used direct algorithms are the filtered-backprojection (FBP) and Feldkamp-Davis-Kress (FDK) [3], and consist of a single, fast filtering operation and a subsequent application of F^T, the adjoint of the X-ray transform. Iterative algorithms often have better noise-reducing properties than direct algorithms, but require multiple applications of F and F^T, which increases the computational cost and time considerably [4].

A (Deep-Learned) image processing or analysis task, such as a denoising algorithm, takes a reconstructed volume $\mathbf{x}_t$ as input, and produces an output $\mathbf{y}_t$. The output can be an image, but can also be a different quantity, such as a vector field or scalar, depending on the image task. We will denote the analysis algorithm with the mapping $A\colon \mathbf{x}_t \mapsto \mathbf{y}_t$. In the forthcoming sections, we will explain RECAST3D, a pipeline for reconstruction and visualization, and choose an image-to-image Deep Learning component for A. In a real-time tomographic pipeline, the processes of *Experimentation* until *Visualization/automation*, i.e., the different components of Fig. 3.1, are taking place concurrently, using software buffers, in order to be efficient. As a result, multiple reconstructions or analysis outputs can be generated from the same projections at time t. For brevity, we will reuse the subscript t amongst $\mathbf{f}_t$, $\mathbf{x}_t$, and $\mathbf{y}_t$.

3.2.2 Experimental set-up with RECAST3D

High-resolution 3D image reconstruction poses a difficulty for real-time applications. To compute a full volume at the potential resolution of the data, algorithms take several minutes of computation time, even when modern hardware and efficient direct solvers are considered. To enable visualization and analysis at much faster frame rates, RECAST3D, a software for real-time reconstruction [12], limits the reconstruction to a few user-selected slices through the volume. The RECAST3D graphical user interface is shown in Fig. 3.2. RECAST3D uses the FBP and FDK algorithms, which, thanks to their linearity and computational structure, allow reconstructing a region of interest with a computation time that is linearly related to the number of voxels in the region. The result is called a *quasi-3D* reconstruction, and

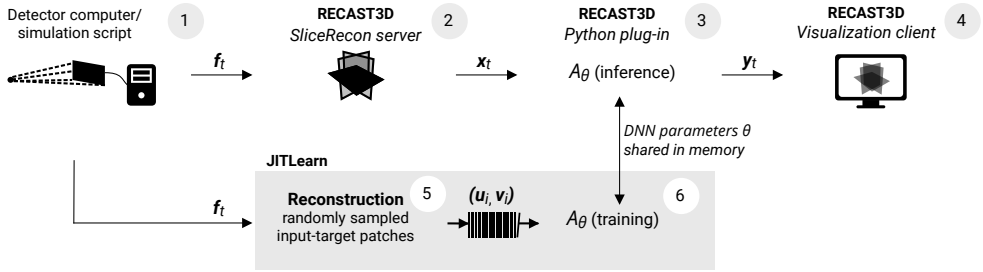


Figure 3.3: The proof-of-concept set-up for just-in-time learning with RECAST3D, using a U-Net for image denoising with the Noise2Inverse loss. **Labels 1 to 4**: The RECAST3D pipeline reconstruct slices $\mathbf{x}_t$ from a stream of projections $\mathbf{f}_t$, and sends these to the visualization client (cf. Fig. 3.2). **Labels 5 and 6**: Our *JITLearn* software intercepts the slices, via a plug-in, at (3). In a concurrent process, the U-Net architecture A_θ is trained on reconstructions $(\mathbf{u}_i, \mathbf{v}_i)$, using a capacity queue (cf. Fig. 3.6). The figure is described in more detail in the main text of the section Software set-up.

permits a refresh-rate of milliseconds. In its graphical user interface, a user may add, remove, and reposition slices, which allows for a fast interrogation of the object during the experiment. RECAST3D has been used for real-time alignment [92], explorative imaging [9], and visualization of experiments with quickly evolving dynamics [93]. In our software configuration, we send the output of the analysis operator, $\mathbf{y}_t$, to the graphical user interface, using RECAST3D's package-based communication protocol. Reconstruction is therefore not restricted to slices, i.e., $\mathbf{x}_t$ could also be a 3D image. The analysis operator, A, on the other hand, must always return a slice. By default, RECAST3D initializes with three orthogonal slices (Fig. 3.2). For the ease of discussion, however, we will limit our analysis to a single slice. The RECAST3D pipeline is illustrated in the top four processes of Fig. 3.3. We will discuss its technical details in the section Software set-up.

For our experiments, we take a subset of projection data from the two dynamic imaging experiments [9], each corresponding to a five-minute interval. The laboratory set-up consists of a polychromatic conebeam source and a Dexela1512NDT CMOS detector. Measurement data is acquired by rotating the sample at a speed of 1.8 seconds per rotation. With an exposure time of 12 milliseconds per frame, this yields 150 projections for each full rotation. The field of view was cropped to the sample holder, which resulted in a 647-by-768 pixel image. Flat and dark field images were collected before the start of the experiment. For reconstructions, we used a voxel size of 0.25 mm and the geometry parameters that are provided with the data. Reconstructions are computed from full rotations (150 projections) with the FDK algorithm, which is a type of filtered-backprojection for the conebeam geometry. For spatio-temporal reconstructions, the interval between neighbouring reconstructions is fixed to 0.720 seconds (60 projections).

3.2.3 Software set-up

For the proposed just-in-time learning strategy, reconstruction software and Deep Learning software need to work closely together. In Fig. 3.3, we illustrate our proof-of-concept set-up. The arrows from (1) to (4) describe the RECAST3D pipeline, which is built on top of its *TomoPackets* library, i.e., a set of software interfaces for passing projections, geometries, reconstructions and user interactions via Push/Pull and Publish/Subscribe protocols in ZeroMQ [97]. Real-time reconstruction requires two processes: one to compute $\mathbf{x}_t$ at fixed projection intervals in the RECAST3D *SliceRecon* server, component (2), and another to continuously generate training data $(\mathbf{u}_i, \mathbf{v}_i)$, at (5). Deep Learning requires two similar processes. The first, a Python script at component (3), accepts $\mathbf{x}_t$, loads θ from memory, and subsequently computes $\mathbf{y}_t = A_\theta(\mathbf{x}_t)$ and sends it to an external machine running RECAST3D. Component (6) accepts a batch of $(\mathbf{u}_i, \mathbf{v}_i)$, performs a training step, and updates the stored parameters θ. In an optimal set-up, the four processes run concurrently, each with a single or multiple GPUs.

In the proof-of-concept software, *JITLearn*, that we release together with this article (see Additional Information), we separate training, i.e., we only run component (5) and (6), and evaluate the trained network at a later time. To simulate a real-time experiment, we preload all projection data to RAM (*Random Access Memory*), and synchronize all GPU processes using a projected experiment start time. Projection data is subsequently released to the GPU processes on the basis of a virtual framerate, which we set to 24 milliseconds per frame to prolong our experimental data from 5 to 10 minutes. Dark and flat field removal, as well as FDK filtering steps, are not computed immediately, but processed on the GPU as soon as required by a reconstruction. During training, we store the network parameters at regular intervals. This allows restoring and evaluating the network on different $\mathbf{x}_t$ afterwards.

The neural network architecture we employ for our experiments with Noise2Inverse is a U-Net [38]. This is a widely adopted architecture for image processing, consisting of a symmetric downscaling (encoder) and upscaling (decoder) part complemented by skip connections. In our PyTorch implementation, which replicates the original U-Net proposal [38], downscaling is performed with strided convolutions, and upscaling with transposed convolutions. Our U-Net has implemented three down and upscaling levels. Each level has three encoding and three decoding convolutional layers, with a concatenating skip connection bridging the two parts. Every level doubles the number of feature maps while halving their resolution, starting with 8 feature maps after the first level. A higher number of levels increases the size of the network's *receptive field*, i.e., the region in the input that, after a sequence of layers, contributes to the determination of a feature. Our implementation accommodates to reconstructions that are *2D*, *2D + time*, or *3D*. Both *2D + time* and *3D* are implemented with 3-dimensional convolutions, rather than a concatenation in the channel dimension. We thus treat a third dimension similar to the first two. We refer to this architecture without modifications as our *Standard* architecture.

For training, we employ the *Adam* optimizer with a learning rate of 0.0001 and batch size of 32. We do not train on full slices, but reconstruct only small regions-of-interest in

the volume, which we refer to as *patches*. The patches are sampled uniformly at random from a slice as well as its spatial neighbourhood in the fluid container. No patches are sampled outside the glass container to prevent the sampling of reconstruction artefacts. While the patch-sampling region is adjustable during the experiment, we currently do not provide a graphical user interface to do so – by default, the optimizer samples from the entire reconstruction volume. For brevity, we will not introduce notation for this sampling process in our analysis or results.

3.2.4 Training data generation

Generating sufficient amounts of reconstruction data poses a key challenge when training in parallel with the ongoing experiment (cf. component (5) in Fig. 3.3). A naive approach would call a reconstruction software for each $\mathbf{u}_i$ and $\mathbf{v}_i$, and with that repeat projection preprocessing steps, as well as CPU-GPU memory transfers, unnecessarily. Obtaining $(\mathbf{u}_i, \mathbf{v}_i)$ as random samples from a single high-resolution volume is not possible either, as high-resolution 3D reconstruction takes seconds to minutes to complete [12]. This would result in either throttling the training process, or introducing a delay before the network can access new reconstruction data.

This problem has motivated the development of a new software package, which we released as an extension to the GPU-accelerated reconstruction framework called ASTRA Toolbox [80]. Our software takes the Nvidia CUDA kernels from the ASTRA package through *CuPy*, a Python interface to the Nvidia CUDA library. The software enabled an implementation of the FDK algorithm that is optimized for repeated reconstructions from a streaming buffer of projection data. Before entering the buffer, projections are pre-filtered with the Ram-Lak filter and stored in *textures*, a CUDA memory structure for efficient interpolation. Modifications to the CUDA kernels furthermore enable efficient backprojection from arbitrary projection subsets, which benefits the efficiency of the Noise2Inverse algorithm.

Figure 3.4 shows run-times of the developed software for differently sized reconstructions. With *slices* we denote 2D reconstructions, with *slabs* thin reconstruction volumes, and with *sequences* three consecutive reconstructions. A small training patch takes about 2 milliseconds, regardless of size, as the computational cost of the reconstruction algorithm for very small reconstructions is dominated by geometry computations. Timings of DNNs for different input sizes display an expected quadratic trend in n_x, i.e., a linear relation to the total number of voxels. For training with small patches, reconstruction is the bottleneck. A batch of 60-by-60 reconstructions, for example, takes 64 milliseconds, while an iteration of the optimizer for these inputs takes 10 milliseconds. During inference, on the other hand, the DNN is typically the bottleneck. Inference on a 1000-by-1000-by-3 slabs takes about 45 milliseconds (22 Hz), and reconstruction is an order of magnitude faster.

3.3 Just-in-time Learning

We will now sketch a context for the just-in-time learning paradigm by formulating three interlinked research topics. The first topic formalizes the continuously varying data that is

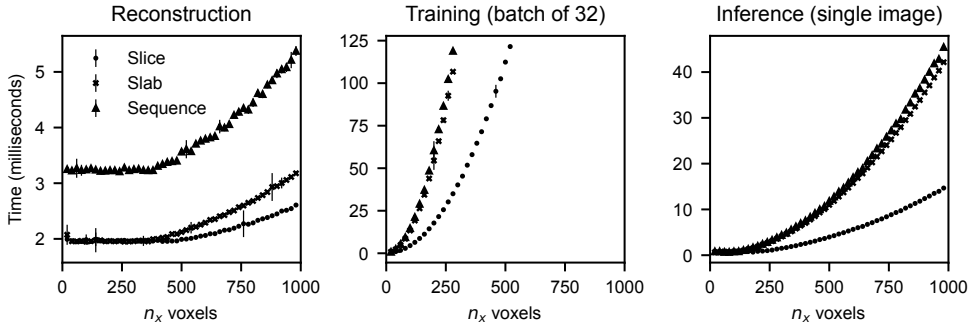


Figure 3.4: Timings, in milliseconds, for n_x-by-n_x slices (•), n_x-by-n_x-by-3 slabs (×), and three consecutive slices (▲). DNN timings use the *Standard* architecture. Training time includes backpropagation, whereas inference time only measures the forward pass. Timings are calculated using the mean over 200 repetitions after a 200 warm-up iterations on an Nvidia GeForce RTX 2080 Ti Rev. A.

encountered during an experiment. In the second topic, we discuss the DNN architecture and its relation to the reconstruction operator. In the third topic we explain the buffer-based online training approach that we developed for pipeline data. We will take the assumption that no previous training data, or ground truths, are available prior to the experiment.

3.3.1 Topic I: Real-time imaging data

In a real-time scientific imaging experiment, reconstructions are often originating from a continuous (physical) process. A data set $\mathcal{D}$ hence consists of consecutive reconstructions. The real-time setting is in that regard different to typical neural network applications, where the data set is often assumed to consist from i.i.d. (image) samples of a distribution $\mathbb{P}$ over $\mathcal{U} \times \mathcal{V}$. Instead, real-time samples are often highly temporally correlated, and describe a type of directed random walk on the manifold of $\mathcal{U} \times \mathcal{V}$. We will see that this is both a "curse and a blessing": while these samples may not appropriately cover the sought data distribution, short sequences of correlated data contain local information that can help neural networks with the analysis task (discussed in Topic II).

A more formal modelling of real-time data in the tomographic pipeline would describe $\mathcal{D}$ as a single realization of a continuous stochastic process with the Markov property [98]. Every input-target pair $(\mathbf{u}, \mathbf{v})$ from $\mathcal{D}$ can thus be interpreted as a sample from a time-dependent $\mathbb{P}_t$ over $\mathcal{U} \times \mathcal{V}$ that is correlated with all previous samples. The evolution of $\mathbb{P}_t$ reflects the evolution of the physical system that is imaged: If the system is in a (possibly dynamic) equilibrium, $\mathbb{P}_t$ remains constant over time (a *steady-state* regime). While subsequent samples are still temporally correlated, a long collection of samples will eventually approximate the distribution. In *transient* regimes, $\mathbb{P}_t$ changes smoothly with t, and each sample is from a slightly different distribution. Transient regimes occur at the beginning of an experiment, during experiments with evolving dynamics, or, for instance, during object

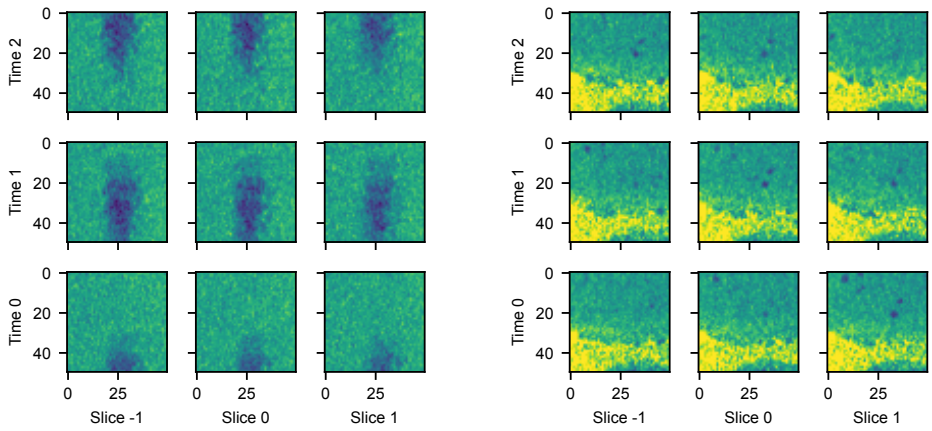
exploration [9] and zooming [90]. They pose a bigger challenge for learning. Obtaining sufficient samples to cover a particular $\mathbb{P}_t$ might require multiple realizations of the same process, i.e., repetitions of the experiment under the same conditions.

However, $\mathbb{P}_t$ may also change abruptly in time, e.g., if the user changes the scan geometry or alters a reconstruction parameter during the experiment. In RECAST3D, a re-orientation of a slice effectively changes the geometry of the reconstruction, and the images may show differently-oriented features of the object. We call these *discontinuous jumps* between continuous regimes. Draws from $\mathbb{P}_t$ at a time t and its next timestep, at $t+1$, could possibly display an unseen part of an object, and therefore radically change the latent distribution of the data samples. A network trained on a data set $\mathcal{D}$, with samples drawn until $\mathbb{P}_t$ at time t, is thus principally unprepared to generalize to samples after a jump, say a data set $\mathcal{E}$ drawn at time $t+1$, as they are not from the training distribution. In Results I, we therefore quantify the discontinuous jumps, and in Results III, we investigate the ability of a network to recover from a discontinuity.

3.3.2 Topic II: Network architectures

In order to function in a just-in-time setting, DNN architectures need to be comparatively small. That is, they need to consist of a limited number of (parameterized) operations, such as layers and channels. The first reason is that, in order to keep up with the incoming data from the tomographic pipeline, the network optimizer must be able to update network parameters quickly enough, which is only possible when the number of operations is small. The second reason is that, during inference, a network is presented images in high resolution. For large detectors, for example, a reconstructed slice at the potential resolution of the data can be as large as 2000-by-2000 pixels. Large networks typically demand too much GPU memory and cannot easily attain high framerates at these resolutions. On the other hand, small networks have limited expressivity, which makes a careful architectural design (e.g., number of hidden layers, residual connections) more important. For image tasks with low spatio-temporal complexity, such as segmentation and denoising, small networks yield satisfactory results [34].

The next consideration for an architecture is what additional reconstructions it should use as inputs to process a given image slice. Neighbouring slices and additional timesteps – usually given to the network as additional image channels – often provide important contextual information, which could help to improve the image task. Fig. 3.5 shows this by example, where, in a region of interest, Experiment (A) has spatially coherent features (along rows), while Experiment (B) has spatio-temporally coherent features (along diagonals). For a denoising task, where redundant patterns contain additional information, a network for Experiment (A) may benefit from a sequence of nearby slices, whereas a network for Experiment (B) improves with a spatio-temporal volume. More generally, this also points to an intricate connection between reconstruction and analysis operators (G and A). Multiple spatial or temporal reconstructions may improve the network performance, but also decrease the reconstruction speed, and delay training and the end-to-end framerate of the



(a) Experiment (A): Large bubble traveling upwards.

(b) Experiment (B): Smaller bubbles in the bottom.

Figure 3.5: Different dynamics between Experiments (A) and (B), illustrated by the spatio-temporal reconstruction of a vertically-oriented subvolume of dimensions $(50, 50, 3)$. In (a), the dissolvent is soap-based, leading to fast-traveling bubbles of overall large size. In (b), a spatio-temporal pattern appears along the time-slice diagonal, due to the slower and smaller bubbles in the gel-based dissolvent.

pipeline. A network should therefore ideally be designed jointly with all parameters of the reconstruction operator, such as the spatial extent, the choice of filter, or image resolution. We will illustrate the importance of architectural choices in Results II.

3.3.3 Topic III: Online learning

Online learning, or incremental learning, is a strategy in which samples are presented in sequential or chronological order during training [99]. The strategy is typically employed in machine learning applications where data is processed in large volumes, as it allows data samples to be discarded after training. During online learning, network weights are continuously updated to fit the most recently presented samples. As a result, a network that generalizes well to a distribution of recent samples may have diminishing accuracy for older samples – a process known as *catastrophic forgetting*. While this behaviour is undesired for learning a static distribution from a stream of data [100], the strategy suits real-time tomography, since distributions are time-dependent and our goal is to generalize only to the most recently reconstructed image from the pipeline. The field of online learning as of today is still developing, but examples exist in image classification [101] and denoising autoencoders [102].

Just-in-time learning is an application of real-time online learning to dynamic imaging data (Topic I), and can be explained through the temporal evolution of neural network weights θ. Without the real-time aspect, its goal at a time t is to find optimal weights $\theta_t^\star$ from reconstructions $\mathcal{D}_t$ (Eq. (1.10)) obtained by drawing from $\mathbb{P}_t$. Note that, since pairs $(\mathbf{u}_i, \mathbf{v}_i)$

are generated separately from $\mathbf{x}_t$ (Fig. 3.3), such $\mathcal{D}_t$ would ideally be designed *ad hoc* by reconstructing input-target pairs that are expected to help generalization towards $\mathbf{x}_t$. For example, when $\mathbf{x}_t$ is from a *steady-state* regime, $\mathcal{D}_t$ could consist of a large amount of reconstructions sampled from the last m projections $\{\mathbf{f}_{t-m}, \ldots, \mathbf{f}_t\}$ corresponding to the regime. As each time index has a corresponding data set and optimum, the dynamics of the real-time experiment leads to a trajectory $\theta_0^\star, \theta_1^\star, \ldots$ in the parameter space, and in principle we would like our learning process to follow this trajectory as closely as possible.

In practice, the challenge of just-in-time learning is not only to approximate $\theta_t^\star$ at each time t as closely as possible, but also to do this within the time constraints of the experiment. Similar to other online strategies, we achieve this by continuing the training with the previous (suboptimal) weights θ_{t-1} and by appending or updating the data set $\mathcal{D}_{t-1}$ with new samples to form $\mathcal{D}_t$. Thanks to the spatio-temporal continuity of $\mathbb{P}_t$, the difference between most data set pairs is small, and an optimization from θ_{t-1} towards $\theta_t^\star$ can therefore be interpreted as a minimal example of *transfer learning* from $\mathcal{D}_{t-1}$ to $\mathcal{D}_t$. Transfer learning can be significantly faster and a neural network can reuse features as well as low-level statistics when θ_{t-1} is in the *basin* of $\theta_t^\star$ [103]. A basin is a region from the parameter space in which local descend optimization methods, e.g., the commonly-used stochastic gradient descend, would converge towards $\theta_t^\star$. Our approach is different to, e.g., *continual learning* where instead the goal is to preserve the structure of the parameter space, for example through constraints on the weights or gradients, or by replaying training data [104].

In our result section, we train DNNs with reconstructions that are generated chronologically. However, as illustrated in Fig. 3.4, sample generation is often slower than optimization, even with our tailored reconstruction software. This forms a bottleneck: If we would allow only single-time access per sample, i.e., let $\mathcal{D}_t = \{(\mathbf{u}_t, \mathbf{v}_t)\}$, in the way other online learning strategies are designed, the optimizer would not be able to continue training until new reconstructions are generated. We therefore introduce a modification that enables resampling of reconstructions for a short duration. We will let the training data set be time-dependent, i.e., $\mathcal{D} = \mathcal{B}_t$, and practically implement this as a buffering capacity queue, as shown in Fig. 3.6. The buffer $\mathcal{B}_t$ at time t contains the last $N_{\text{buffer}} := |\mathcal{B}_t|$ draws from $\mathbb{P}_t$. By enabling asynchronous drawing from the buffer, the optimizer may select randomly from $\mathcal{B}_t$ while new reconstructions are continuously added.

The size of the buffer, N_{buffer}, is a user-specified parameter. If an optimizer is able to handle n_{opt} samples per second, and reconstructions are generated with n_{reco} samples per second, a sample will on average be picked $n_{\text{opt}}/n_{\text{reco}}$ times during its lifetime in $\mathcal{B}_t$. The choice of N_{buffer} thus does not affect how often a sample is picked. A suitable choice, however, may not be straightforward. A too small buffer may not contain sufficiently diverse features. In contrast, a buffer too large may unnecessarily retain reconstructions that have become irrelevant for the current timestep. In Results III, we will inspect the online learning process for different buffer strategies.

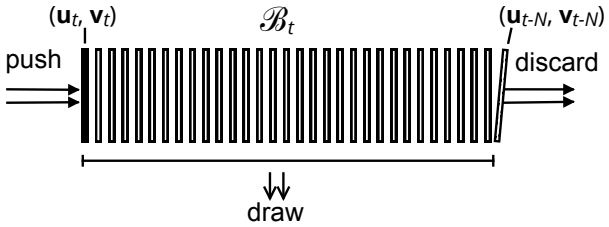


Figure 3.6: Online learning implemented with an N_{buffer}-sized capacity queue $\mathcal{B}_t$. Training-target pairs $(\mathbf{u}_t, \mathbf{v}_t)$ are pushed onto the queue, while the network optimizer draws pairs asynchronously by a uniform random sampling of their indices.

3.4 Results

We investigate the three topics of the preceding section with the Noise2Inverse method as learning task and data collected using our experiments. To quantify results for samples in small time windows, the mean-squared error loss does not provide a robust performance metric, due to the fluctuating noise levels that occur during data collection in our set-up. Similar to the loss and expected risk (equation (1.11)), we therefore define an *accuracy* α and *empirical accuracy* $R^{\alpha}_{\mathcal{D}}$, with

$$\alpha(A_\theta, \mathbf{u}, \mathbf{v}) := \frac{\|A_\theta(\mathbf{u}) - \mathbf{v}\|_2^2}{\|\mathbf{u} - \mathbf{v}\|_2^2}, \quad R^{\alpha}_{\mathcal{D}}\left[A_\theta\right] := \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{u}_i, \mathbf{v}_i) \in \mathcal{D}} \alpha\left(A_\theta, \mathbf{u}_i, \mathbf{v}_i\right). \tag{3.1}$$

In comparison to the mean-squared error (MSE), the denominator of $\alpha(A_\theta, \mathbf{u}, \mathbf{v})$ in Equation (3.1) additionally normalizes by the noise magnitude. As a result, we expect the empirical accuracy to be $R^{\alpha}_{\mathcal{D}} \in [0.5, 1]$. To see this, first consider a network that does not denoise at all, i.e., $A_\theta(\mathbf{u}) = \mathbf{u}$, and note that this leads to $\alpha(A_\theta, \mathbf{u}, \mathbf{v}) = 1$ and $R^{\alpha}_{\mathcal{D}} = 1$. Then, consider the ideal Noise2Noise setting (discussed in Methods) in which $\mathbf{u} = \hat{\mathbf{u}} + \epsilon$ and $\mathbf{v} = \hat{\mathbf{u}} + \epsilon'$, where $\hat{\mathbf{u}}$ is the noiseless image and ϵ and ϵ' denote two independent realizations of voxelwise-i.i.d. noise images with zero mean and finite variance σ^2. A perfect denoising network would predict $A_\theta(\mathbf{u}) = \hat{\mathbf{u}}$, and therefore

$$\alpha(A_\theta, \mathbf{u}, \mathbf{v}) = \frac{\|\hat{\mathbf{u}} - (\hat{\mathbf{u}} + \epsilon')\|_2^2}{\|(\hat{\mathbf{u}} + \epsilon) - (\hat{\mathbf{u}} + \epsilon')\|_2^2} = \frac{\frac{1}{n}\sum_i^n \epsilon_i'^2}{\frac{1}{n}\sum_i^n (\epsilon_i - \epsilon_i')^2}, \tag{3.2}$$

where $n := n_x n_y n_z$ is the number of voxels. For large n, i.e., for typical image sizes, the nominator will be close to σ^2 whereas the denominator will be close to $2\sigma^2$, thus $\alpha(A_\theta, \mathbf{u}, \mathbf{v}) \approx 0.5$. Using the Noise2Inverse setting, however, we will occasionally see that $R^{\alpha}_{\mathcal{D}} < 0.5$. This is possible when sparse-angle artefacts or fast object motion inflicts additional differences between the even-angle ($\mathbf{u}_t$) and odd-angle ($\mathbf{v}_t$) reconstructions, in other words, when $\mathbf{u}_t$ and $\mathbf{v}_t$ have different underlying ground truths. When a DNN network learns to predict such differences, this reduces the numerator $\|A_\theta(\mathbf{u}) - \mathbf{v}\|_2^2$ in Eq. (3.1).

3.4.1 Results I: Real-time imaging data

As discussed in Topic I, a real-time network continuously encounters differently distributed samples, due to transient and discontinuous changes in the data distribution. We call these *out-of-distribution samples*. Compared to the hold-out samples, both types of samples have not been seen by the network before, but the out-of-distribution samples are not distributed like the training set. To quantify the error this induces, we train the *Standard* network A_θ on a data set $\mathcal{D}$, while evaluating it on samples of an unseen data set $\mathcal{E}$. At the same time, we also train a baseline network A_ψ on $\mathcal{E}$. The accuracy change is approximated by

$$\epsilon := R^{\alpha}_{\mathcal{E}}\left[A_\theta\right] - R^{\alpha}_{\mathcal{E}}\left[A_\psi\right], \tag{3.3}$$

i.e., the difference between A_θ, trained on $\mathcal{D}$ but evaluated on $\mathcal{E}$, and the baseline A_ψ, trained on $\mathcal{E}$ and evaluated on $\mathcal{E}$.

Figure 3.7 displays six scenarios for $\mathcal{D}$ and $\mathcal{E}$. Scenarios (a)-(c) correspond to discontinuous changes due to user interaction, and (d)-(e) show the transient regimes during the experiment. Both $\mathcal{D}$ and $\mathcal{E}$ are generated from 12 seconds of projection data of Experiment (A). The error ϵ is the difference between the dashed line and the red dotted markers. For *repositioning*, $\mathcal{D}$ is the top half of the volume, and $\mathcal{E}$ the bottom half. For *tilting*, we change horizontal to vertical slices, and for *zooming* we reduce the voxel size to 0.125. In the second row of the figure, the choices for $\mathcal{E}$ are data sets that are generated from different projections. In (d), projections are taken 12 seconds directly after $\mathcal{D}$. In (e), the data is from 25 minutes later in the same experiment. For (f), 12 seconds of data are taken from Experiment (B).

In Fig. 3.7 most scenarios show that evaluation on out-of-distribution samples cause a significant loss in accuracy, and that training A_θ for a prolonged time does not improve this evaluation. For user interaction, the error ϵ is highest in the case of *zooming*. This is expected, as convolutional operators are not scale-invariant, and therefore take notion of the size of the learned image features. We note that a similar change in the data distribution could also occur when the acquisition geometry changes. In panel (d), training on $\mathcal{D}$ generalizes well to the next 12 seconds. In (e), however, the object dynamics in Experiment (A) have changed substantially, as more smaller bubbles occur in the data. This led to a decreased accuracy. In (f), we find that generalization towards Experiment (B) is poor, likely due to the change in liquids and set-up. We will inspect the visual effect of out-of-distribution evaluation in Results III.

Our results in this section, for Noise2Inverse and Experiment (A), indicate that small DNNs can be sensitive to changes in the pipeline data, i.e., the learned image features from 12 seconds of data are not generic enough to be useful for differently placed, oriented or zoomed slices. While this may suggest extending the training data $\mathcal{D}$ with reconstructions from $\mathcal{E}$, similar to data augmentation, this is not straightforward. Set-up changes or discontinuous changes in the physics cannot easily be anticipated by augmentation. Moreover, small networks may not be expressive enough to anticipate large amounts of data, and the generation of additional data will require more computational resources.

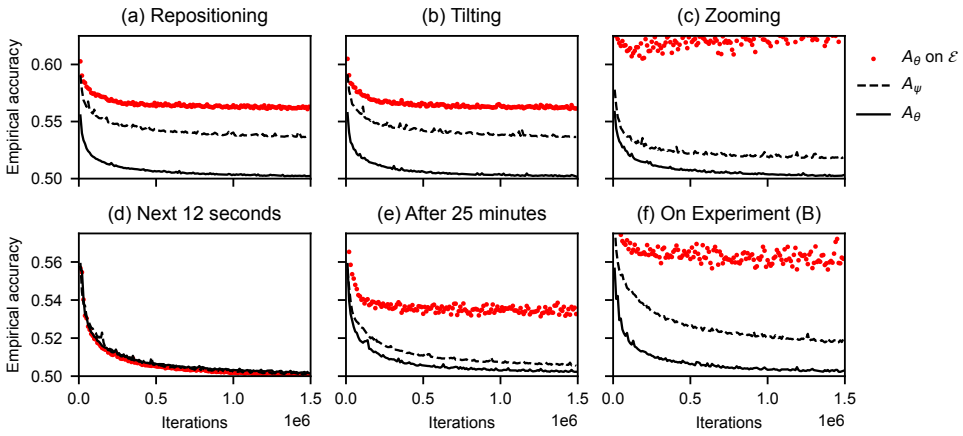


Figure 3.7: Empirical accuracy of a network with parameters θ trained on $\mathcal{D}$, and a network with parameters ψ trained on $\mathcal{E}$. In (a)-(c), $\mathcal{E}$ is a change to $\mathcal{D}$ due to interaction with the experiment, and in (d)-(f), $\mathcal{E}$ is a data set from different projections. The solid and dashed lines denote the accuracies of A_θ and A_ψ on 3,200 hold-out samples. The error of A_θ on out-of-distribution samples, i.e., Eq. (3.3), is the distance between the potential accuracy, i.e., the dashed baseline $R^\alpha_{\mathcal{E}}[A_\psi]$, and the achieved accuracy, i.e., the red markers $R^\alpha_{\mathcal{E}}[A_\theta]$.

3.4.2 Results II: Network architectures

In the following, we explore the performance of different U-Net architectures, and look at the effect of reconstruction dimensions for Experiments (A) and (B). Table 3.1 lists four variations of our *Standard* architecture, the network trained on randomly-selected 20-by-20 input patches. *Cx*-networks use x feature maps after the first layer (and then follow the U-Net principle). With more feature maps the *representational capacity* of the network increases, i.e., the network stores a larger number of filters. The next class, *Vx*-networks, use larger x-by-x-sized patches during training, which increases the receptive field during training. *Zx*-networks use x nearby spatial reconstructed slices, and *Tx*-networks use x timeframes, with the middle frame as target. Both increase the image input during training and inference, and use 3D convolutions. During 30 minutes of training, samples are continuously generated from a fixed 12-seconds data set of projection data, and previous reconstructions are allowed to be resampled by the optimizer to avoid throttling. This is representative of online training in an experiment where the data has no temporal component yet. For evaluation, 3,200 hold-out samples are taken from a 12 second interval later in the experiment.

Figure 3.8 displays the evolution of empirical accuracies for the architectures presented in Table 3.1. The results show that all denoising networks can be trained to a satisfactory accuracy in 10 minutes or less. Training is approximately fast for all networks, although the *Cx*, *Zx*, and *Tx* networks become significantly slower when evaluated on high-resolution slices. This results in a lower framerate at the RECAST3D graphical interface. This is not the case for the *V30–V150* architectures, which used larger patches. While larger patches do

	Feature maps	Input	Frames	n_{opt} (per second)	framerate*
Standard	8	$(20, 20, 1)$	1	4,190	73
C16	16	·	·	4,119	38
C32	32	·	·	4,058	16
V30	·	$(30, 30, 1)$	·	3,725	72
V40	·	$(40, 40, 1)$	·	4,111	72
V60	·	$(60, 60, 1)$	·	2,983	71
V100	·	$(100, 100, 1)$	·	1,303	71
V150	·	$(150, 150, 1)$	·	591	71
Z3	·	$(20, 20, 3)$	·	3,721	29
Z5	·	$(20, 20, 5)$	·	3,820	17
Z7	·	$(20, 20, 7)$	·	3,778	10
T3	·	·	3	4,597	23
T5	·	·	5	4,342	14
T7	·	·	7	3,935	10

Table 3.1: The network architectures in Topic II, corresponding to various parameter choices, that are used to compare accuracy and training speed in Figures 3.8 and 3.9. "·" denotes a *Standard* parameter (see Software set-up). n_{opt} denotes the number of samples per second drawn by the network optimizer during training. *Corresponding to a 1024-by-1024 output slice.

not affect the DNN framerate, we note that the slower initial convergence was only valuable for Experiment (B), where it eventually resulted in a better accuracy.

Figure 3.9 illustrates the best architectures in each category, with an evaluation on a hold-out sample. Overall, the architectures produce comparable outputs, suggesting that convergence speed may be the most important criterion for selecting a network. In Experiment (A) the more expressive network (*C32*) and more slices (*Z7*) help to produce a slightly smoother background. In Experiment (B), *Z7* predicts bubbles better, and *T7* produces the sharpest bubble interfaces, although the latter predicted streak artifacts too. This is likely due to a difference that occurs between even-angle and odd-angle reconstructions in Noise2Inverse. Yet, since the dynamics in Experiment (B) are slower, the temporal network may also be better at using the redundancy of features between multiple frames.

3.4.3 Results III: Online learning

We conclude with a real-time simulation of Experiment (A) by replaying the full projection data set to our just-in-time software using the *T3* network (Table 3.1). When $t < 336$ seconds, reconstructions are taken in random orientations from the top of the fluid container, and when $t \geq 336$ seconds, samples are taken from the bottom. This simulates a discontinuity in the pipeline data, corresponding to the repositioning of a horizontal slice in RECAST3D (panel (a) in Fig. 3.7). In this scenario, the reconstructions were generated at a speed of 9 batches per second, while the *T3* network trained with 141 iterations per second.

Figure 3.10 shows the empirical accuracy of the network during training, estimated with 3,200 samples in a small window Δt around each t, i.e., the accuracy $R^{\alpha}_{\mathcal{D}_{\Delta t}}$. The solid black

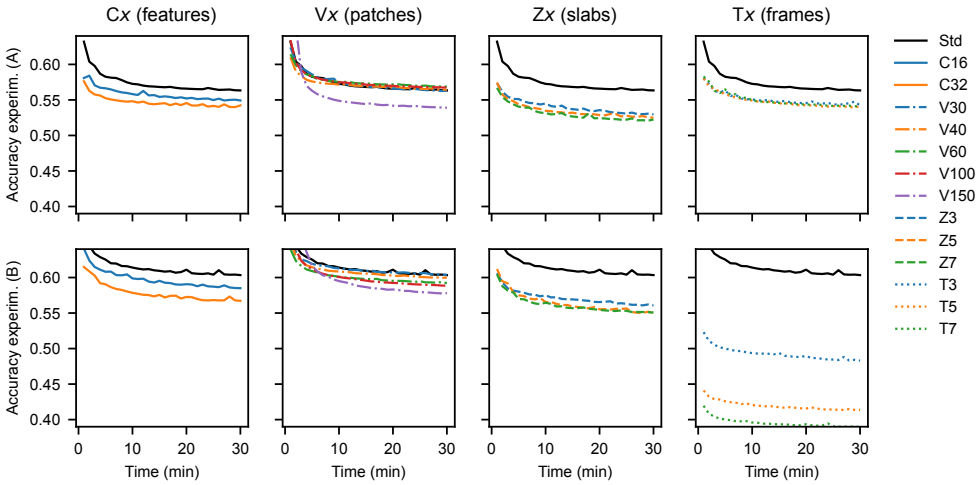


Figure 3.8: Empirical accuracy of the architectures in Table 3.1 for Experiment (A) and (B), using 3,200 hold-out samples per evaluation point. Increasing the features (Cx) or slices (Zx) is effective, although they slow down the DNN framerate. The effect of patch size (Vx) and frames (Tx) is dataset-dependent.

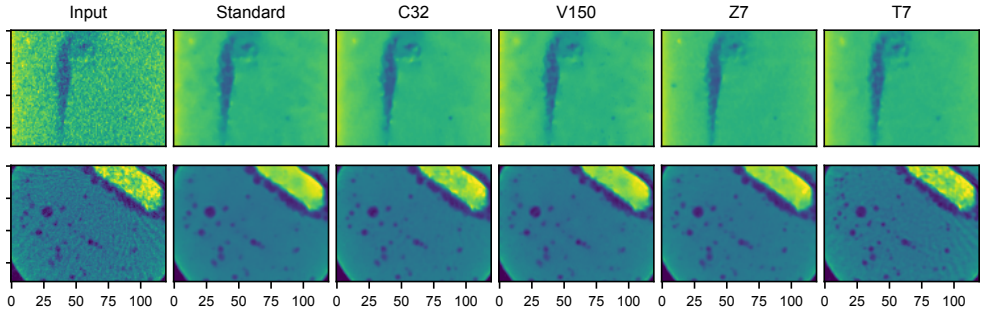


Figure 3.9: 90-by-120 region-of-interest network outputs, comparing the best-performing DNNs in each class of Table 3.1 visually. **Top row**: experiment (A). **Bottom row**: experiment (B). *Standard*, C32, V150, Z7, and T7 outputs, for Experiment (A) and (B), are the best networks from Fig. 3.8 evaluated on *Input*. Experiment (A) shows a rising bubble in a vertically-oriented slice, and Experiment (B) a horizontal cross-section in the middle of the volume.

line marks the accuracy of the *T3* offline-trained network on all data before the discontinuity, the dashed line a Total Variation (TV) based denoising [105] implementation from the Python scikit-image package, and the dotted line describes a training strategy without buffer. The blue and red lines mark the just-in-time strategy with a buffer of 32 and 320 reconstructions, respectively. Our first observation is that both buffer sizes perform better than a network that processes the input sequentially. The network with $N_{\text{buffer}} = 320$ (about a second of data), performs best, which suggests that including previous reconstructions in the training process is advantageous in the case of slow data generation.

Figure 3.11 visualizes images of the pretrained and $N_{\text{buffer}} = 320$ network before and after the discontinuity. We note that at $t = 4$ minutes (first row), bubbles of the just-in-time network are slightly oversmoothed, compared to the pretrained network, but that the results are of satisfactory quality. For TV, the regularization parameter was set to $\lambda = 0.03$ and the number of iterations to $n_{\text{iter}} = 300$, after manually selection using 3,200 samples from the top of the container. In these experiments, TV denoising was not able to outperform the data-driven denoisers, and proved significantly slower than neural network evaluations. In future work, classical TV may be combined with neural networks in the tomographic pipeline, for example as a training target in an auto-differentiation pipeline [106].

When, after 336 seconds, samples are taken from the bottom of the volume (see the tablet in Fig. 3.2), the pretrained and just-in-time strategies observe a similar accuracy. Interestingly, the just-in-time strategy requires only a short time to improve over the pretrained network. The second row in Fig. 3.11 shows, the strategy at $t = 8$ minutes, has a clear visual advantage. It maintains the fine-grained structure in the tablet (yellow), and is better able to remove the noisy background in the gas around the tablet (dark blue). A video, combining Figures 3.10 and 3.11 for the $N_{\text{buffer}} = 320$ network, is included with the Supplementary Information in the publication associated with this chapter.

3.5 Discussion

Recent advances in Deep Learning have produced powerful image processing and analysis algorithms that operate in a fraction of the time that conventional algorithms take. Our *just-in-time* concept brings these algorithms to real-time tomographic pipelines at synchrotron light source facilities and X-ray microscopy laboratories, where a large diversity of imaging data is processed. The leading principle of just-in-time learning is that continuous regimes in the experiment lead to spatio-temporal continuous data in the pipeline, which can be be used to train while the experiment is ongoing. By combining small, easy-to-train architectures with online learning, the approach allows the analysis of reconstructions during the experiment, and can therefore be employed without preparation and without prior data. In this article, we have demonstrated just-in-time learning using the quasi-3D reconstruction paradigm proposed in RECAST3D [12], and for the small U-Net architectures [34, 38] on the Noise2Inverse loss [40]. For extending it to high-resolution fully-3D reconstructions, fast DNN inference poses the main challenge. While image partitioning techniques can divide the workload over multiple GPUs [107, 108], such techniques have not yet been demonstrated

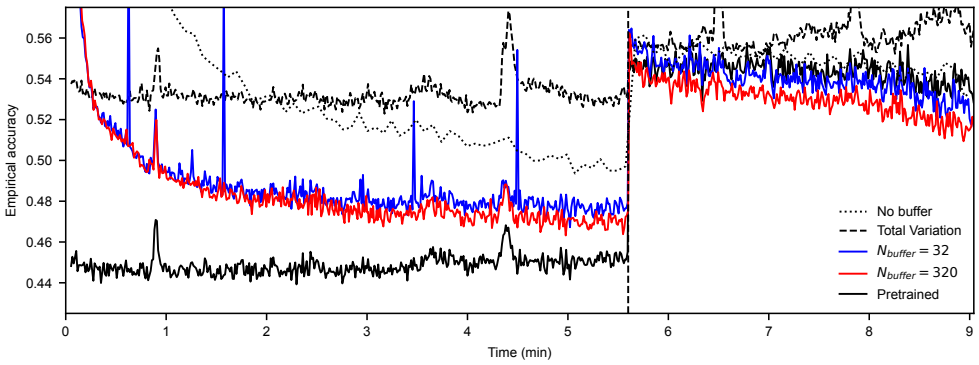


Figure 3.10: Accuracy of the online-trained *T3* network on Experiment (A), with different buffer sizes, in comparison to a pretrained baseline, Total Variation based denoising, and a training strategy without buffer. After 336 seconds, reconstructions are sampled from the bottom of the fluid container to simulate a discontinuity in the pipeline data. Each point on the graph is an evaluation of the network on 3,200 randomly selected samples.

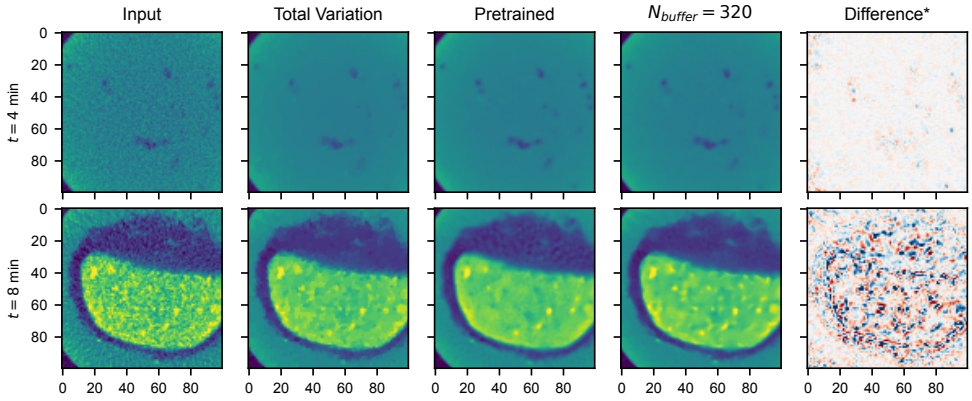


Figure 3.11: A comparison between the input image to denoise, denoising results of TV minimization, and pretrained and just-in-time trained network after $t = 4$ and $t = 8$ minutes, i.e., before and after the discontinuity of the experiment in Fig. 3.10. *The difference is taken between the pretrained and $N_{\text{buffer}} = 320$ outputs.

on millisecond timescales.

In Results I, small networks are found to be sensitive to changes in the data distribution, even within a single experiment. This is generally unavoidable, since, as Results II illustrates, more expressive networks are significantly slower. At the same time, Results I and II show that reconstructions have a large degree of spatio-temporal continuity when the experiment is in steady-state or transient regimes. This enables spatio-temporal DNNs, and allows a neural network to generalize to data from the tomographic pipeline. In Results II, several hyperparameter settings, such as the number of feature maps, led to good visual results. However, we did not discuss how such hyperparameters are to be picked for a new set-up, task, or experiment. An interesting direction for future work would be to find heuristics to automate this. For Results III, we showed a straightforward buffer strategy for online learning. Yet, it is also possible to design the buffer or training data set adaptively. Constraints on gradients, or specialized network architectures, are further possibilities to retain information over longer time spans of the experiment [104]. This, and a more precise description and exploration of the formal concepts behind just-in-time learning that we sketched in Topic I and III, are important directions of future research.

We have demonstrated the feasibility of our approach using real-world laboratory data on an image denoising task using the Noise2Inverse training loss. While we expect this to be representative of other tasks with low spatio-temporal complexity, such as image super-resolution, segmentation, or artefact removal, future research is needed to see what is achievable for more sophisticated image processing tasks – especially when more diverse image features need to be stored in the network. Strategies such as dynamic pruning [109, 110], more recent (e.g., U-Net, transformer) architectures [111], mixed precision weights, and parallelization can improve the speed and accuracy of DNNs, which would widen the applicability of our approach by enabling more expressive networks. For denoising, *just-in-time* learning already proves to be a viable Deep Learning paradigm with a large potential use. Deep Learning-based analysis can provide valuable feedback during an experiment, which can be used for experimental control [92] or real-time scan adaptation and optimization techniques. For the latter, we will investigate coupling the proposed just-in-time learning strategy with reinforcement learning approaches [112].

Chapter 4

ASTRA KernelKit: GPU-Accelerated Projectors for Computed Tomography using CuPy

This chapter is based on the article: Adriaan Graas, Willem Jan Palenstijn, Ben van Werkhoven, and Felix Lucka. "ASTRA KernelKit: GPU-accelerated projectors for Computed Tomography using CuPy". In: *Applied Mathematics for Modern Challenges* 2.1 (2024), pp. 70–92. DOI: 10.3934/ammc.2024004

4.1 Introduction

Computed Tomography (CT) is an imaging technique utilized in scientific, medical and engineering disciplines to resolve the 3D interior of an object from a series of 2D projection measurements. In the most commonly used scanning geometry, *conebeam CT*, X-ray source and detector describe a circular path around the object. However, CT also exists for other imaging modalities and acquisition geometries. For instance, neutron or proton tomography can offer a different contrast than X-ray CT, and in electron microscopy, an electron beam generates projection images from nano-scale objects that are tilted over a limited range of angles inside a vacuum. The wide range of imaging scenarios leads to a vast body of tailored data-processing and reconstruction algorithms described in the literature [4].

Most CT image reconstruction algorithms make use of at least one of two common building blocks: The *forward projection* (FP) maps the object to simulated projection images by modeling the physics and geometry underlying the data acquisition. The model is typically the linear X-ray transform. The *backprojection* (BP) is the adjoint of the forward projection, and maps projection images into the reconstruction volume. The ASTRA Toolbox provides software implementations of these building blocks, and is specifically designed to facilitate the implementation and development of CT algorithms for non-standard geometries [23]. The framework is open source under the GPLv3 license, written in C++, and co-developed by *Vision Lab* at the University of Antwerp and *Centrum Wiskunde & Informatica* (CWI) in Amsterdam. ASTRA Toolbox has been used in multiple tomographic disciplines, e.g., ptychography [114] or neutron imaging [115], and serves as back-end in several frameworks [25, 26, 116, 117].

Due to their pivotal role within algorithms, projection operators – commonly referred to as "projectors" – constitute a significant research focus within computational imaging, inverse problems and scientific computing [4, 7, 118, 119]. The operators process the scan geometry, e.g., a conebeam or parallel beam source, and circular or helical orbits, to emulate X-ray (back)projection lines. The precise algorithmic formulation thereof, which entails, for instance, the discretization and interpolation in the volume, critically influences the accuracy and efficiency of reconstruction. Also computational aspects of projectors are of research interest [120]. To handle large datasets effectively, projectors need to employ memory-efficient strategies, such as partitioning reconstruction volumes into manageable chunks. Moreover, they frequently distribute tasks across multiple computing devices.

Traditionally, ASTRA Toolbox users only invoke projectors indirectly, via calls to one of the pre-implemented CT algorithms (e.g., FBP, FDK, CGLS or SIRT [4, 5, 6, 23, 121]). Figure 4.1(a) illustrates this process. In the first step, a user passes input data, such as the sinogram and geometry, to the algorithm. This is possible due to a Python-C++ interface leveraging the Python-to-C compiler called Cython. In the case of 3D data, the input is then directed to a projector based on NVIDIA CUDA. This is a parallel computing platform and C++ programming model for Graphical Processing Units (GPUs). The final computation is then performed in a CUDA *kernel*, which is a relatively small C++ function that runs on many GPU cores (further discussed in Section 4.2.1).

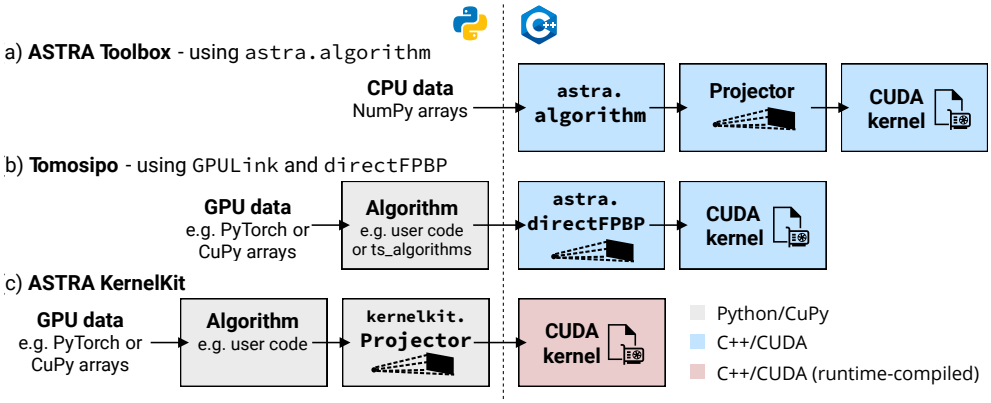


Figure 4.1: Software overview of ASTRA Toolbox, Tomosipo [26], and ASTRA KernelKit, illustrating where its components are located with respect to the Python-C++ language barrier. (a) and (b) show two methods of accessing a projector in ASTRA Toolbox. In comparison, the new ASTRA KernelKit uses a Python-based projector, as well as runtime compilation of the CUDA kernel. The figure is further explained in Section 4.3.1.

In the past decade, algorithms and data pipelines in CT applications increasingly adopted GPU acceleration for tasks beyond (back)projection. Notable are *machine learning* (ML) and *deep learning* (DL) – data-driven algorithms that are leading in the field of inverse problems in imaging [29, 30, 31]. As a consequence, there was a need to pass GPU data more efficiently between external Python frameworks (e.g., PyTorch or CuPy) and ASTRA projectors. In response, ASTRA Toolbox introduced two functionalities. The first, `GPULink`, imports *GPU arrays* (*tensors*, in machine learning terminology) from external packages into ASTRA Toolbox. GPU arrays are multi-dimensional Python arrays that are backed by GPU memory but can directly be manipulated in Python. The second functionality, `directFPBP`, executes CUDA projectors on such external data. When combined, they streamline the use of projectors, and enhances data exchange efficiency with external frameworks. Tomosipo [26] leverages this functionality in a package with a more intuitive Python front-end for geometry manipulations and visualization. Figure 4.1(b) illustrates how this approach gave Python users more flexibility.

Recent advances such as the development of more advanced detector hardware, data transfer, and storage capabilities, have enabled the acquisition of ever larger and more detailed X-ray CT data sets [122], for both laboratory X-ray systems and synchrotron light source facilities. As a result, GPUs are becoming mainstream hardware components for high-performance computing, and frequently used in data preprocessing pipelines. New algorithms, on the other hand, can no longer only be tested on small data sets or toy problems anymore, but instead require a tight integration with GPU-accelerated data processing tools. Scientists and developers working on CT algorithms would therefore benefit from a more fine-grained control over CUDA projectors. However, due to the Python-C++ language barrier, the functionality of CUDA projectors has remained difficult to access up to now.

Recent versions of CuPy and NVIDIA *CUDA Python* have addressed this difficulty, by unifying Python and CUDA into a single interface, which provides full coverage of the low-level CUDA functions. Based on these developments, we present *ASTRA KernelKit*: new high-level CUDA projectors that are fully accessible in Python. In Figure 4.1(c) it is illustrated that the fundamental operations inside the X-ray projection and backprojection, such as data operations or algorithmic formulations, have now either shifted over the language barrier or can be compiled during the Python script. This enables more flexible algorithm development as well as debugging in Python. In this paper, we will explain these advantages and showcase the approach for patch-based reconstructions for deep learning, real-time CT algorithms, and kernel tuning. The article first revisits CT and its implementation in the case of a conebeam geometry in Section 4.2, then provides an overview of KernelKit in Section 4.3, and demonstrates its use for the aforementioned applications in Section 4.5.

4.2 Tomographic reconstruction

4.2.1 The ASTRA CUDA conebeam backprojector

In ASTRA Toolbox, projectors are implemented using CUDA [123]. Their implementations involve two main tasks. Firstly, projectors manage the reconstruction volume, projections, and set-up geometry, including handling transfers between the CPU and GPU device. Secondly, they perform computations associated with the line integrals described in Eq. 1.5. For the latter, building the projector $\mathbf{A}$ as a (sparse) matrix in memory is generally inefficient or impractical. The matrix-vector products $\mathbf{A}x$ or $\mathbf{A}^T y$ that most algorithms need are instead computed *matrix-free*. This entails implementing a function that, e.g., directly computes $\mathbf{A}x$ given x as input.

Kernels are CUDA functions that execute in many parallel threads on GPU cores [123]. A GPU consists of several *streaming multiprocessors* to launch these threads. Threads are divided into *thread blocks*, and each thread computes a small portion of the parallel task. For CT, this portion corresponds to processing a small part of the volume or projection data. In ASTRA Toolbox, kernels are implemented for 2D and 3D conebeam and parallel beam geometries. In this article, we focus specifically on the 3D conebeam backprojection, using the previously mentioned voxel-driven approach, which is most commonly used in high-performance and experimental-data scenarios. However, the concepts presented can be applied to the other projectors in the same way.

The backprojector is summarized in Algorithm 1 using color-coded loops to distinguish CPU code, CUDA parallelization, and kernel code. The main CPU loop processes the projection data y by launching a kernel for each subset of $\bar{N}_\theta$ projection angles sequentially. Each launch issues a sufficient number of 2D $\bar{N}_x, \bar{N}_y$-sized thread blocks to cover the input volume. A single thread block is responsible of a fixed-size 3D $\bar{N}_x, \bar{N}_y, \bar{N}_z$-subvolume of x (cf. the green `for` loops). The partitioning of the entire reconstruction problem is therefore defined by four parameters $\bar{N}_x, \bar{N}_y, \bar{N}_z, \bar{N}_\theta$ (cf. line 1). In ASTRA Toolbox, they are compiled with constant values 16, 32, 6, 32 into the binary code. These values are taken after a

Algorithm 1: The ASTRA Toolbox CUDA conebeam backprojection $\mathbf{A}^T$.

Input : Volume $x \in \mathbb{R}^{N_x, N_y, N_z}$, projections $y \in \mathbb{R}^{N_\theta, N_u, N_v}$, source and detector geometries.
Output: An updated $x \in \mathbb{R}^{N_x, N_y, N_z}$.

```
1  const int Ñ_x, Ñ_y, Ñ_z, Ñ_θ ← 16, 32, 6, 32          ▷ ASTRA Toolbox defaults
2  TextureObject ỹ ← ToTexture(y)                        ▷ Using a CUDA Array
3  • for θ_start in {θ_start Ñ_θ : 0 ≤ θ_start Ñ_θ < N_θ} do
4      • for z_start in {z_start Ñ_z : 0 ≤ z_start Ñ_z < N_z} parallel do   ▷ Blocks
5          • for i in [0 .. N_x − 1] parallel do          ▷ Ñ_x threads
6              • for j in [0 .. N_y − 1] parallel do      ▷ Ñ_y threads
7                  • for k in [z_start .. z_start + Ñ_z − 1] do
8                      • for l in [θ_start .. θ_start + Ñ_θ − 1] do
9                          (p, q) ← ProjectRay(source, detector, i, j, k, l)
10                         x_{i,j,k} ← x_{i,j,k} + Interpolate(ỹ_l, p, q)
```

Algorithm 1. The ASTRA Toolbox CUDA voxel-driven conebeam backprojector algorithm $\mathbf{A}^T$. The algorithm sequentially processes subsets of $\bar{N}_\theta$ angles (line 3), and volumetric chunks of $\bar{N}_z$ vertical voxels in parallel thread blocks (line 4). The •-loops correspond to CPU code, •-loops to CUDA parallelization, and •-loops to kernel code.

bruteforce search over multiple geometries and volume sizes. In Section 4.5.3, we will explore different selections of parameters using a kernel tuning strategy.

Zooming in onto lines 7–10 of Algorithm 1 reveals the idea of the voxel-driven approach. Rather than tracing rays backward though the volume, a single kernel thread processes a stack of $\bar{N}_z$ voxels and collects contributions from $\bar{N}_\theta$ projections. Doing so, GPUs can leverage the locality of projection data by *coalescing* write operations to the reconstruction volume, i.e., by combining the writes of multiple threads to adjacent memory locations into a single memory transaction [20, 120]. Moreover, handling multiple voxels in a thread can reduce geometry computations and benefit from instruction-level parallelism [124]. For each voxel $x_{i,j,k}$ and projection angle l, a thread first computes the intersection of an X-ray at a point $(p, q) \in \mathbb{R}^2$ in the detector plane, using the spatial coordinates of the X-ray source and voxel. It then calculates a contribution to $x_{i,j,k}$ via linear interpolation of the pixel values around (p, q) in y_l. This is efficiently performed using *CUDA textures*. Textures are specific read-only structures in CUDA that enable fast and accurate bilinear or trilinear interpolation in 2D or 3D arrays. ASTRA Toolbox creates such 3D texture, before the kernel call (line 2), with a *3D CUDA Array* as the underlying resource. A CUDA Array is a data structure in which elements are stored with better spatial locality than linear memory. We will further discuss these data structures in Section 4.5.3.

4.3 Software

4.3.1 ASTRA KernelKit

KernelKit, our extension to ASTRA Toolbox, is designed to enhance the accessibility of tomographic projectors, like Algorithm 1, for scientists working within the Python ecosystem. This facilitates prototyping of new algorithms and customization for high-performance use cases. KernelKit maintains the capability to work with nonstandard geometries, a prominent feature in ASTRA Toolbox.

To illustrate its advantages, we compare KernelKit to the most flexible approach that is currently possible with ASTRA Toolbox, see Figure 4.1(b). This approach, which is the working principle of Tomosipo, consists of two parts. The first, `GPULink`, is a straightforward method to point ASTRA to a contiguous memory region on the GPU allocated by an external software (e.g., a PyTorch tensor). In contiguous memory, all bytes are stored sequentially, to allow efficient memory access. The second, `directFPBP`, offers a forward or backprojection that directly outputs into this external region. The combination enables efficient use of ASTRA Toolbox projectors, while allowing data manipulation with external frameworks. KernelKit expands and improves upon this functionality in three crucial aspects:

- *Python-based projectors*: Except for the kernels, KernelKit projectors are fully written in Python. In comparison to ASTRA Toolbox, there is no Python-C++ interfacing code, and projectors are transparently accessible. The user is therefore in full control of the lifetime of the projectors as well as the data that it stores internally. For example, KernelKit geometries can be modified without destroying and recreating the projector. Application-specific tomographic modalities, e.g., hyperspectral or dynamic CT, or algorithms with partial data, e.g., the random-batch gradient descent, often require a fine-grained control over the projectors to be most efficient.

- *High-level CUDA availability*: The linkage between KernelKit (Python) and NVIDIA CUDA (C++) is provided by CuPy. CuPy is a multi-dimensional array library for GPU-accelerated computing with Python [125]. As a drop-in for NumPy and SciPy, it implements a wide range of algorithms and linear algebra utilities using CUDA-libraries. Since KernelKit is built upon CuPy, users can leverage advanced CUDA functionality within and alongside custom algorithms and projectors. Section 4.5.1 will illustrate an example involving the use of CUDA graphs. In contrast, developing CUDA-accelerated algorithms with ASTRA Toolbox is less straightforward due to the language barrier that prevents accessing CUDA data types, such as streams or texture objects, created in the toolbox.

- *Runtime compiled kernels*: KernelKit compiles CUDA kernels during the Python script, via the NVIDIA runtime compiler (NVRTC). In contrast to ASTRA Toolbox, which uses precompiled kernels, variables such as the $\bar{N}_x$, $\bar{N}_y$, $\bar{N}_z$, $\bar{N}_\theta$, or the processing axis order, can now be decided during the Python script. With runtime compilation,

an existing kernel can be swapped out for a user-written implementation of the Radon transform, or an implementation can be optimized on basis of the input data.

4.3.2 Package overview

Since our software is written on top of numerical libraries, it benefits from concise formulations of the mathematical operations. This keeps the source code of our package to a minimum, while providing an expressive and flexible interface. The code consists of three levels:

1. *CUDA kernels*: FP and BP kernels are implemented as CUDA source files. As kernels are runtime compiled, the end-user can provide custom sources. Additionally, we use Jinja2 [126], a placeholder-style templating engine, to generate a specialized CUDA kernel code before compilation. The voxel-driven conebeam kernel, for example, is stored in `cone_bp.cu`.

2. *Kernel classes*: A Python class parses, compiles, and calls the CUDA sources, e.g., `VoxelDrivenConeBP` handles the `cone_bp.cu` kernel. These classes abstract low-level CUDA tasks, such as the configuration of block sizes, precomputation of kernel parameters, and passing of arguments to the kernel. They furthermore do not perform CPU-GPU memory transfers. For custom kernels or specialized algorithms, the user can extend or modify their behavior.

3. *Projectors*: The last layer abstracts the details of the kernel and provides an interface for algorithms at the level of $\mathbf{A}$ and $\mathbf{A}^T$. A projector, such as `ForwardProjector` or `BackProjector`, compiles a kernel on initialization, and executes it multiple times on invocation. Projectors provide a detailed interface for memory management, axes conventions, and geometries, and are the recommended tool for custom algorithms.

Additionally, a few auxiliary modules provide the tools to create and manipulate geometries, e.g., `ProjectionGeometry` and `VolumeGeometry`. As examples for algorithms, we provided the Feldkamp-Davis-Kress algorithm [6] as `fdk()`, and SIRT [4] as `sirt()`. `XrayTransform` furthermore builds a linear X-ray operator on top of the projectors, in the same style as SciPy operators and the MATLAB Spot toolbox [127]. This enables a simpler interface, and is more suitable for a high-level integration with other packages.

4.3.3 Related software

In this section we position KernelKit by contrasting it with related software solutions. We restrict the comparison to packages that implement CUDA projectors, and that expose these via a Python front-end. They are ASTRA Toolbox, TIGRE [24], Pyro-NN [128], TorchRadon [129], and TomocuPy [130]. Other commonly used frameworks are Tomosipo [26], Operator Discretization Library (ODL) [117], TomoPy [25], and Core Imaging Library (CIL) [116]. Rather than implementing projectors, they use ASTRA Toolbox, TIGRE and/or Scikit-image as a back-end, and provide additional algorithms, mathematical abstractions, or data processing functionality on top.

We first acknowledge that KernelKit does not aim to provide extensive algorithmic functionality, like TomoPy, ODL, CIL, TIGRE or ASTRA Toolbox. However, its distinctive aspects, i.e., Python projectors, high-level CUDA, and runtime compilation (Section 4.3.1), are currently not offered by any other package. Another unique aspect about KernelKit is its inherited support for *vector geometries*, meaning that the source and detector do not need to describe a perfect circular orbit for (back)projection.

In terms of performance, we show that our conebeam backprojection matches or exceeds ASTRA Toolbox, depending on the configuration of our package (see Appendix 4.4). Since Tomosipo uses ASTRA Toolbox as a back-end, this also holds for Tomosipo. Several packages provide alternative backprojection or reprojection operators. For example, TorchRadon provides a multi-channel 2D backprojection kernel, and TomocuPy implements a kernel for GridRec reconstruction [131]. These kernels are understood to outperform ASTRA Toolbox counterparts for the use cases that they are suited for. However, the strength of KernelKit lies in its flexibility, as it allows users to easily add or customize kernels according to their requirements.

All compared packages are on the same level as Tomosipo (Fig. 4.1(b)), meaning that the main software features are behind the language barrier. Yet, there are also benefits to approaches with C++ projectors. In ASTRA Toolbox and TIGRE, projectors and algorithms can be shared with MATLAB. In Pyro-NN and TorchRadon, projectors are written as TensorFlow C++ and PyTorch C++ extensions, and subsequently exposed to Python as neural network layers. In these approaches, the projectors can better integrate with the low-level features of the machine learning framework. While Python projectors enable easier debugging and development, C++ is sometimes better-suited for low-latency or highly multi-threaded environments.

Among the compared packages, the design principle behind TomocuPy is closest to KernelKit. TomocuPy utilizes both Python, with CuPy, and C++, using C++-level CUDA libraries. TomocuPy does not provide Python objects for projectors or kernels, but is aimed at high-performance reconstruction pipelines for synchrotron light sources. In their approach, the two languages are connected via a thin Python-C++ interface called SWIG (Simplified Wrapper and Interface Generator) [132].

Lastly, package developers should consider the convenience of integrating projectors into their application. ASTRA Toolbox's direct API, illustrated in Fig. 4.1(b), offers straightforward projector invocation. Tomosipo [26] enhances this with a more intuitive Python front-end for geometry manipulations and visualization. On the other hand, TIGRE provides functions in Python to invoke $\mathbf{A}x$ or $\mathbf{A}^T y$, but lacks support for GPU arrays, making integration less straightforward. KernelKit, TomocuPy, Pyro-NN and TorchRadon all use tensor or array data structures from their host frameworks (CuPy, TensorFlow and PyTorch). These leverage DLPack, a standardized approach for exchanging GPU arrays.

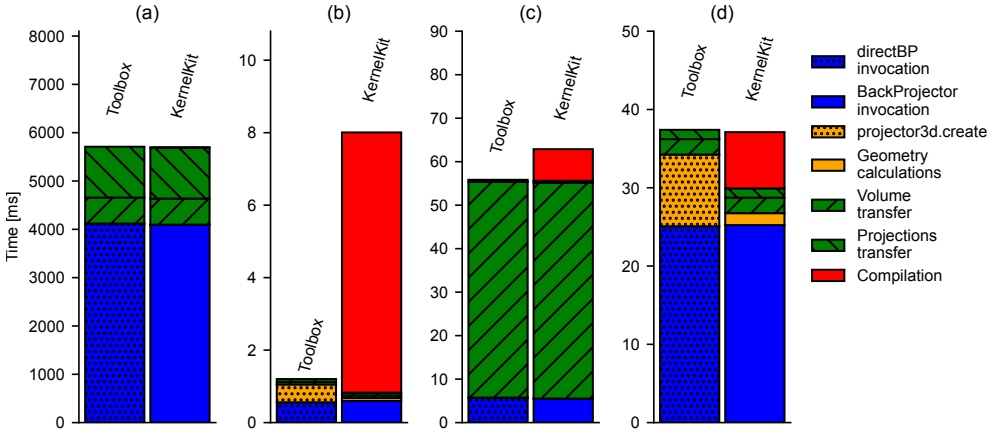


Figure 4.2: Validation of KernelKit in "Toolbox configuration", i.e., such that it precisely matches the behavior and kernel settings of ASTRA Toolbox, on an NVIDIA GA102GL [RTX A6000]. (a) is a 1000^3 volume, (b) a 50^3 volume, (c) a $(300, 300, 1200)$ volume with $N_\theta := 3$ projections of size $(300, 1200)$, and (d) is a 2000-by-2000 slice reconstructed by $N_\theta := 1000$ images of a $(2000, 1)$ line detector.

4.4 Runtime and computational overhead analysis

In the following, we configure KernelKit in "Toolbox configuration", i.e., such that it precisely matches the behavior and kernel settings of ASTRA Toolbox. We subsequently compare the runtimes of different components of the two software packages side-by-side. This validates that our Python implementation is correct, and allows us to detect computational overhead differences (e.g., due to the use of Python, CuPy, or the NVRTC compiler). Moreover, it verifies that the comparison between the two software packages is fair. We take the ASTRA Toolbox conebeam kernel with default parameters, and disable the use of advanced CuPy features. Software tests are used to validate the implementation numerically. Note that the comparison does not aim to demonstrate the best achievable runtime of either package.

Figure 4.2 displays a break-up of calls to ASTRA Toolbox, using `directFPBP` (cf. Section 4.3.1), and KernelKit, using `BackProjector`, for four reconstruction problems. The problems are selected such that they load the projector differently. Problem (a) and (b) are a large and small reconstruction problem. Problem (a) consists of a 1000^3 volume, 1000 angles, and a 1000-by-1000 detector. Likewise, problem (b) has a 50^3 volume, 50 angles, and a 50-by-50 detector. Problem (c) is a reconstruction from a severe sparse-angular geometry ($N_\theta := 3$), and problem (d) is a 2D slice geometry ($N_z := 1$ and $N_v := 1$).

Figure 4.2(a)-(d) show that `directFPBP` and kernel execution in our package (blue segments) have virtually equivalent runtimes, confirming that we can use ASTRA Toolbox as a

baseline measurement. In (b), the smaller reconstruction, we find that geometry processing in Python is typically slower than in C++, but that in some cases ASTRA Toolbox projector initialization is somewhat slower (orange). This overhead is in many contexts negligible, as both frameworks allow the reuse of projectors. For algorithms that use a single backprojection, such as the filtered-backprojection or FDK (Section 4.5.2), a considerable amount of time may be spent in data transfers between the host (CPU) and device (GPU). With the ASTRA Toolbox algorithms API, Fig. 4.1(a), such transfers are unavoidable. However, using Tomosipo, Fig. 4.1(b) or KernelKit, data can be kept on the GPU.

Figure 4.2(c) shows that for a set-up with three angles, only a small amount of time is spent in the kernel execution, compared to the transfer of the resulting volume. In Figure 4.2(d) the geometry is reduced to a slice: in this case ASTRA Toolbox switches to dedicated kernel with $\bar{N}_z := 1$ (cf. the standard value of $\bar{N}_z := 6$ in Algorithm 1). Also in this scenario, we observe a runtime of KernelKit that is similar to the ASTRA Toolbox precompiled binary (blue segments). This suggests that runtime compilation does not lead to observably better or worse performing binary code. Lastly, the compilation of a kernel with CuPy (red) takes a few milliseconds. This is insignificant for most applications, since the kernel is cached after its first use. Yet, it can be of importance for kernel tuning (Section 4.5.3), where each kernel configuration requires a recompilation.

4.5 Case studies

In the upcoming sections, we will demonstrate KernelKit through practical examples. While we are aware of many, often quite complex, applications that could benefit from customized projectors, particularly in dynamic imaging and machine learning domains, we have chosen three studies that showcase KernelKit's core advantages (Section 4.3.1) while being simple to describe and easy to interpret. The scenarios use the previously-introduced conebeam backprojector in reconstruction contexts that are inspired by experiments from our FleX-ray laboratory in Amsterdam [9] and the X-ray imaging set-up for fluidized beds at Delft University of Technology [46].

The implementations of our case studies require customized projectors, high-level CUDA features, or runtime-compiled kernels. They highlight the complexity of real-world scenarios, which often pose challenges that cannot be effectively addressed from the Python interface of ASTRA Toolbox alone. As an alternative to solving these in C++ library code, which would request significantly more development time from the average Python user, we will present tailored solutions with KernelKit. In each case study, we will compare the KernelKit solution to the most efficient use of ASTRA Toolbox, which is the direct projection approach illustrated in Fig. 4.1(b). This is also equivalent to Tomosipo [26].

For a fair comparison, we have ensured that KernelKit projectors can be configured to reproduce the performance of ASTRA Toolbox (Appendix 4.4). This allows us to explain and attribute performance gains to the enhancements that we describe. In the experiments, we will use ASTRA Toolbox 2.1.2 with CUDA Toolkit 11.7; KernelKit with CuPy 12.1 and CUDA Toolkit 12.1; PyTorch 2.1.0.dev20230821 with CUDA Toolkit 12.1; and a mod-

ified version of Kernel Tuner 0.3.0. All the experiments use Python 3.10 and the NVIDIA GA102GL [RTX A6000] architecture.

4.5.1 Patch-based neural network training

In this case study, we will configure a KernelKit projector for reconstruction of image *patches*, i.e., small 2D or 3D image regions. Such a projector is particularly useful for neural network training, where CNN parameters are optimized based on a large data set consisting of pairs of input and desired output. Training on patches that are, e.g., reconstructed from random locations in a 3D volume, offers several advantages over training on high-resolution inputs. For example, it can mitigate GPU memory exhaustion and reduce training time. This approach has proven successful in various image tasks such as super-resolution and denoising [133, 134].

To avoid slowing down the training process, a large quantity of patches must be readily available. When patches are only required as network inputs, one option would be to precompute and store a vast number of patches on disk, which can then be loaded back during training. This is, however, only a practical strategy for relatively small data sets. In dynamic CT, or when neural networks use reconstruction as a layer, on-the-fly generation of the patches from projection data in parallel to the training process can be faster and more resource-efficient. During online learning, generating patches ahead of the experiment may not be feasible, and patches must be generated in parallel to the neural network training process [86].

Since patches can be as small as 20-by-20 voxels, even small computational overheads in the projector have relatively large impact on the overall backprojection time. Recognizing that a projector is executed repetitively, while the geometry and data sizes remain constant, enable an alternative implementation that is better suited to this specific use case. In the following, we will show that we can identify and reduce two overheads, and tailor KernelKit conebeam backprojectors $\mathbf{A}^T$ (Algorithm 1) to improve upon the ASTRA Toolbox baseline (Fig. 4.1(b)).

We will use Figure 4.3 to display timings from the implementations on example data. The data consists of $N_\theta := 1000$ images of an $(N, 1)$ resp. (N, N)-sized detector in the 2D, resp. 3D, case. The input size is varied from $N \in \{50, 100, \ldots, 500\}$ for a 2D patch, and $N \in \{20, 30, \ldots, 80\}$ for a small 3D volumetric input. We implement each backprojection algorithm as a PyTorch neural network layer, and time the execution of the forward pass through the layer. In all ASTRA Toolbox and KernelKit implementations, geometry calculations are not part of any timings, as they can be avoided by recycling an existing projector object (i.e., the orange segments of Figure 4.2 in Appendix 4.4).

ASTRA Toolbox We first show baseline implementations of ASTRA Toolbox (Fig. 4.1(a) and (b)), i.e., the ASTRA 3D backprojection algorithm, and the ASTRA Toolbox approach taken by Tomosipo. In the former, data is kept in host memory and passed back-and-forth to the GPU, while in the latter, data is kept on the GPU using `GPULink`. Figure 4.3 shows the results with green and black squares, respectively. The results emphasize that a careful

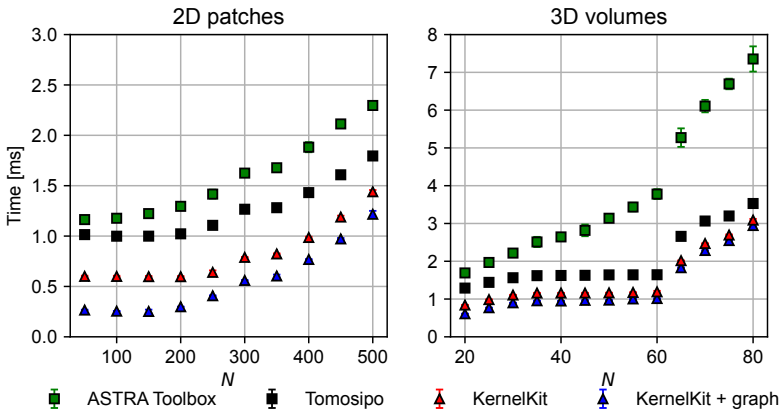


Figure 4.3: Conebeam backprojection PyTorch layers for a $(N, N, 1)$-sized input (left) and (N, N, N)-sized volume (right), using $N_\theta := 1000$ projections of an $(N, 1)$ and (N, N)-sized detector. 2D timings use 2000 samples, and 3D timings 350 samples. Both cases use GPU warm up with an equivalent amount of burn-in samples.

integration between ASTRA Toolbox and external software is necessary to obtain good performance. When, for example, a neural network layer follows after the backprojection, e.g., a PyTorch convolution on the GPU, an unnecessary "ping-pong" occurs by a device-host-device transfer. We note that, like Tomosipo, KernelKit retains volumes and projections on the GPU, as transfers are explicit commands in the CuPy framework.

KernelKit A first KernelKit implementation, denoted by red triangles in Figure 4.3, shows the timings of a PyTorch layer based on a `BackProjector` instance. In this reconstruction context, with a fixed geometry, it is advantageous to instruct the KernelKit projector to recycle as many memory operations as possible. Firstly, previously-created CUDA textures can be recycled, meaning that they can be overwritten with new projection data when the dimensions of the reconstruction problem are unchanged. This eliminates memory reallocation and the creation of new texture objects. More precisely, in Algorithm 1 line 2, the object $\tilde{y}$ is not destroyed or recreated on every projector invocation. While this would generally come at the disadvantage of occupying additional memory in-between projector calls, this does not pose a problem due to the small memory size of patches. Secondly, a re-transfer of geometry parameters from global GPU memory into *constant memory*, i.e., a fast read-only memory that is shared by all threads, can be avoided. This is about 47 kilobyte for 1,000 angles, approximately the size of a 100-by-100 patch. Fig. 4.3 shows that these optimizations result in a consistent decrease in reconstruction time of about 0.3 ms per patch.

KernelKit + CUDA graphs In the left part of Figure 4.3, we note that at the smaller dimensions, $N < 250$, reconstruction times do not decrease further towards zero, in both KernelKit and ASTRA Toolbox. This is an indication that a part of the computation is not

yet fully explained by the cost of backprojection in the first implementation. A recent CUDA feature, called *CUDA Graphs*, provides a mechanism to launch multiple CUDA kernels with a single CPU operation. This reduces the overall launch time of kernels, and can be used to avoid the recomputation of kernel arguments on the CPU side. Using the integration between CuPy and PyTorch, we construct a CUDA Graph in PyTorch (i.e., a PyTorch Graph) by capturing all kernel launches after a warm-up iteration of the neural network. In a next iteration, with unchanged geometries, the graph replays all the kernel launches using the same parameters, but with the newly-placed data in memory. Figure 4.3 shows, with the blue triangles, that the CUDA graphs are able to remove some of the overhead, and reduce the backprojection time to approximately 0.25 ms. We conclude that patches can be generated about four times faster than the optimal implementation with ASTRA Toolbox, which does not allow an integration with CUDA Graphs due to CUDA data structures not being able to pass through the language barrier. The results from this use case can be combined with kernel customization, which we discuss in Section 4.5.3, to achieve even faster patch generation.

4.5.2 Real-time X-ray tomography

Reconstruction software generally presumes a common workflow where a set of projection images is provided by the user and where the desired output is a single reconstruction volume. Algorithms for less conventional contexts, e.g., with multiple output volumes, or with a subset of projection images as input [26], often require modifications to the low-level algorithm primitives in the software to be implemented in the most efficient way. In this section, we will demonstrate an example from *real-time tomography*, where changing the backprojector in the FDK algorithm (Eq. 1.6) improves the efficiency of reconstruction. Real-time tomography is used to visualize dynamically evolving processes in synchrotron light source facilities and X-ray microscopy laboratories [9, 34, 90]. Live feedback, provided by the reconstructions, simplifies steering of the experiment as well as the on-line optimization of acquisition settings [93]. With that it prevents experimental repetition and therefore saves valuable experiment time.

In real-time tomography, the processes from data acquisition to reconstruction are streamlined with *tomographic pipelines* [92]. Pipelines consist of sequential software components, which can be distributed over multiple machines or GPUs. After data is measured at the detector, it is first preprocessed, e.g., with dark-field and flat-field corrections, and linearization according to Beer-Lambert's law [5]. For FDK, a filtering step is subsequently performed (i.e., the convolution $y \circledast \mathbf{f}$ in Eq. 1.6), and the final step is to apply the conebeam backprojector to the filtered data in order to reconstruct the object (Algorithm 1). The duration of data processing tasks is linear in the size of the projection data $\mathcal{O}(N_\theta N_u N_v)$, whereas reconstruction is $\mathcal{O}(N_x N_y N_z N_\theta)$ using our voxel-driven conebeam kernel, or, alternatively, $\mathcal{O}(N_x N_y N_z \log N_\theta)$ in the case of GridRec [131]. As each pipeline component processes an individual chunk of the pipeline data asynchronously, the visualization framerate follows the bottleneck component, which is often the backprojection. The performance of the real-time

FDK in a tomographic pipeline therefore depends on the GPU-accelerated backprojection component.

Although modern GPU technology has pushed the boundaries of what is achievable in parallel computing, high-resolution fully-3D reconstruction and visualization in milliseconds is still out of reach [12, 130]. While reducing the spatial resolution or visualization framerate would render the problem feasible, it is often desirable to reconstruct at the potential resolution of the set-up, i.e., such as defined by the detector resolution and exposure time, in order to follow the imaged physics. One successful approach to do so is the *quasi-3D reconstruction*, which was first introduced in the real-time reconstruction pipeline called RECAST3D [12]. The principle of a quasi-3D reconstruction is that only a few user-selected cross-sectional slices through the volume need to be reconstructed (see Figure 3.2 in chapter 2). The RECAST3D graphical user interface enables interaction with the slices, and allows for a fast interrogation of the object during the experiment. The FDK algorithm is especially well-suited in this situation, as it enables a region-of-interest reconstruction in a time that is linearly related to the number of voxels in the region. RECAST3D has been used for real-time alignment [92], explorative imaging [9], and visualization of experiments with quickly evolving dynamics [93].

A straightforward quasi-3D implementation with ASTRA Toolbox or Tomosipo, using the approach in Fig. 4.1(b), is to reconstruct the three (orthogonal) cross-sectional slices individually, and send each of the slices sequentially to the RECAST3D user interface. This requires three ASTRA Toolbox projectors, each configured to handle one slice of the quasi-3D reconstruction. However, this approach leads to unnecessary repetitions in converting projections to CUDA texture objects ($\tilde{y}$ on line 2 of Algorithm 1), which is required for fast interpolation in the sinogram (Section 4.2.1). With KernelKit projectors, on the other hand, users can capitalize on the fact that the same projections are used in each projector. First, a single texture object $\tilde{y}$ can be constructed, and kept in memory for as long as the projector is needed. This avoids memory reallocation, similar to Section 4.5.1. Then, $\tilde{y}$ can be shared with the three KernelKit projectors. As a result, only a single $\tilde{y}$ has to be updated on incoming data to RECAST3D, saving memory and computation time. The optimizations have the potential to significantly lower the quasi-3D algorithm runtime, as slice reconstructions, with complexity $\mathcal{O}(N_x N_y N_\theta)$, have the same order of complexity as data tasks, $\mathcal{O}(N_\theta N_u N_v)$.

We will compare the KernelKit projectors to the ASTRA Toolbox approach on three set-ups that are used for dynamic CT in the scientific literature. The first is our FleX-ray laboratory micro-CT scanner at CWI [9]. The second is a recent high-speed rotational set-up that achieves a half-rotation reconstruction every 10 ms [135], and the third used a high-resolution detector to study the rheology of liquid foams at the SLS TOMCAT beamline [17, 122]. We will time the KernelKit projector using an average of 400 quasi-3D reconstructions after 400 warm-up samples. Note that, in comparison to Section 4.5.1, in this section we consider the geometries (i.e., the slices) to be variable, and require the projection data to remain constant during the reconstruction of the slices. In Section 4.5.1, the geometries were

instead constant, and the projections were variable.

Table 4.1 lists the framerates that are achievable with quasi-3D reconstruction for the three set-ups. We note that for the FleX-ray micro-CT set-up, the ASTRA Toolbox baseline already achieves a quasi-3D backprojection framerate of 4.8 ms, which is faster than the framerate of the detector (12 ms). For the High-speed set-up, a reconstruction can be obtained once every 3.5 half-rotations (180°), and about one time per half-rotation for the Tomobank set-up, although in the latter case the framerate of 3 Hz is comparatively slow. Compared to ASTRA Toolbox, the optimizations with KernelKit lead to a factor 8–18 speed-up. In summary, Table 4.1 shows that customization of quasi-3D backprojection can lead to significantly faster visualization framerates and enables a reconstruction in each half-rotation in three set-ups. Yet, reconstructing at the framerates of modern detectors, for example for the purpose of automation, remains an open challenge.

	FleX-ray [9]	**High-speed** [135]	**Tomobank** [17, 122]
Detector framerate	12 ms (83 Hz)	78.125 µs (12.8 kHz)	0.8 ms (1.3 kHz)
Detector resolution	400×600	1024×1024	2016×1800
Slice resolution	400×400	1024×1024	2016×2016
Projections per 180°	75	128	300
180° rotation time	900 ms (1.1 Hz)	10 ms (100 Hz)	210 ms (4.8 Hz)
Toolbox	4.8 ms (208 Hz)	35.8 ms (28 Hz)	291.7 ms (3 Hz)
KernelKit	**0.6 ms (1.6 kHz)**	**1.8 ms (0.6 kHz)**	**15.7 ms (64 Hz)**

Table 4.1: Quasi-3D backprojection for three simulated set-up configurations. Preprocessing, uploading or filtering are not part of any timings.

4.5.3 Kernel optimization using Kernel Tuner

In the last use case, we will leverage KernelKit's ability to runtime compile kernels with CuPy and the NVIDIA runtime compiler (NVRTC), to search over CUDA parameters and kernel implementations. This is termed *kernel tuning*, and entails maximizing the performance of GPU computing, by optimizing free parameters of kernels, such as block sizes and algorithmic constants [124]. Tuned algorithms achieve better runtimes, reduced energy consumption [27, 28], or utilize less resources, in particular GPU memory. In high-throughput applications, such as in-line CT scanning, a kernel can be tuned towards a fixed measurement protocol and dedicated GPU architecture. In these situations, even a slight improvement can lead to significant energy savings over the equipment's lifetime.

The efficiency of backprojection depends, in the first place, on the GPU architecture and the dimensions of the reconstruction problem (i.e., N_x, N_y, N_z, N_u, N_v and N_θ). Natural targets for tuning are the CUDA *thread block sizes*, i.e., free parameters that define the number of threads grouped together within a single thread block for parallel execution on the GPU. This is constrained by the capability of the CUDA architecture; commonly 1024 threads per thread block, and set to sensible defaults in ASTRA Toolbox. Another target for tuning is the backprojection code itself, known as *software tuning*. In Algorithm 1, for

example, the compiled $\bar{N}_z$ and $\bar{N}_\theta$ constants, which define how the problem is chunked, can be made variable again through the process of recompilation. In our software, CUDA/C++ kernel code is parameterized through the Jinja2 templating engine [126]. In this way, different code paths can be explored through recompilation at the program runtime.

To demonstrate the potential of kernel tuning, we demonstrate three tuning results, using KernelKit in conjunction with the Kernel Tuner software package [136]. We utilize the KernelKit projector in such a way that the sinogram and volume are retained on the GPU, allowing millisecond kernel recompilation and testing for each point in the parameter search space. Kernel tuning with ASTRA Toolbox or Tomosipo would require a scripted recompilation of ASTRA Toolbox, which would be significantly slower. Moreover, such alternative would have less flexibility in exploring templated code paths, as ASTRA Toolbox is built around fixed axis and geometry conventions. We furthermore note that our results are specific for the NVIDIA RTX A6000 architecture, and that these results do not necessarily generalize to reconstruction problems of different dimensions, e.g., with low or high numbers of angles or non-standard volume geometries.

Figure 4.4 displays the result of a bruteforce search over all possible CUDA thread block sizes for the reconstruction of a large, 2000-by-2000 voxels, slice. The projection data has dimensions $N_\theta := 32$, $N_u := 2000$, $N_v := 2000$. We picked $N_\theta = \bar{N}_\theta$ to time a single kernel launch of Algorithm 1. We warm-start the GPU for every configuration with 50 samples, and average over 100 subsequent samples. For the timings, we launch CUDA graphs to eliminate CPU overhead, and eliminate the cost of all data transfers by preloading data onto the GPU. The search takes about one hour to complete, and finds a minimum at $(16, 4)$. The associated runtime of 0.397 ms is about 8% faster than that of the uninformed standard choice of $(16, 32)$, which yielded 0.431 ms. We confirm that this is a valid optimum by repeating the $(16, 4)$ and $(16, 32)$ configurations several times with 100 samples. We did, however, note a slow decrease of the speed-up with increased numbers of averages. Yet, after 40,000 averages, the new optimum nevertheless leads to a stable improvement of 3% over the uninformed default choice. Such an excessive load may, however, not be representative of real-world usage of the kernel.

To demonstrate searching over implementations, we parameterize the conebeam kernel to allow backprojection from four different texture memory back-ends (cf. line 1 in Algorithm 1), which are termed *resources* in the CUDA specification [123]. The first option is a texture object with a *3D CUDA Array* (the default used by ASTRA Toolbox, see Section 4.2.1). A 3D CUDA Array can be used with any axis order, and can therefore avoid an in-memory transposition of the data. The second option uses a *Layered CUDA Array*, which is optimized for spatial look-up in the second and third dimension, and is often more efficient when N_θ is the major array axis. The third option uses a list of N_θ two-dimensional texture objects, rather than a single texture. Compared to a layered texture, this enables a partial update of projections in-between kernel invocations, for example in a dynamic imaging scenario with a moving timewindow of projections. The last option uses N_θ texture objects backed up by linear memory, which avoids the creation of a CUDA Array. Here, the projections are stored

in *pitched* arrays, meaning that the minor array axis is padded to a multiple of 32 for faster look-up.

To compare the four options, Figure 4.5 runs Kernel Tuner for increasingly larger reconstruction problems. Figure 4.5(a) displays backprojection with an *initialization* of textures, and Figure 4.5(b) a backprojection with an *update* to existing projections. Figure 4.5(a) shows that, for algorithms that require a single invocation of $\mathbf{A}^T$, such as the FDK algorithm, layered CUDA arrays yield the best result, thanks to their fast initialization. Figure 4.5(b) shows that linear memory is only marginally slower than layered arrays, and that their main disadvantage originates from the slow initialization of N_θ texture objects. Textures that use pitched linear memory are therefore an alternative to CUDA Arrays when an algorithm requires write access to y, or when GPU memory is scarce.

Kernel tuning holds a large potential for in-line and industrial CT, and particularly for scientific imaging equipment, as optimized CUDA kernels can improve algorithm runtimes or reduce energy costs. In [46], a set-up for ultrafast imaging of bubbling fluidized beds was introduced at *Delft University of Technology*. The set-up consists of three stationary X-ray sources and flat panel detectors. In this last example, the detectors operate in $(300, 1548)$-pixels regions of interest at 65 Hz. Each 3-tuple of frames provides the sparse-angular projection data that is necessary to compute a $(300, 300, 1200)$-sized reconstruction volume. To find an optimal backprojection kernel, we use the bruteforce search strategy in Kernel Tuner with a search space consisting of texture options and the $\bar{N}_z$ parameter (Algorithm 1). Kernel Tuner finds that a configuration consisting of layered CUDA arrays and $\bar{N}_z := 2$ is optimal, and that this improves the runtime from 3.90 ms (ASTRA Toolbox default parameters) to 2.93 ms, an improvement of 25%. Using the found parameters, an optimization over block size multiples of 8 finds that $(152, 1)$ further improves the runtime to 2.18 ms, a 44% improvement compared to the defaults. As fluidized bed experiments comprise several minutes of experimentation, and may contain several thousands of timeframes, tuned kernels realistically improve the efficiency and costs of reconstructing bubbling fluidized beds.

We note that there are many facets of kernel tuning for X-ray CT that we have not explored in this article. For large reconstruction problems, for example, searching over the parameter space takes increasingly more time, and a bruteforce search strategy may therefore not be feasible. Timing a single kernel that is representative of the entire reconstruction problem may then be able to provide a solution. Another topic of further research is to find small search spaces that can be explored quickly before the start of an iterative algorithm. This could already trade off in a faster runtime, even when the algorithm is ran once. In a follow-up study we will explore the geometry-dependence of the kernels as well as the application of the different optimization strategies in Kernel Tuner to X-ray CT.

4.6 Conclusion

ASTRA KernelKit is an all-Python CT reconstruction package that leverages the ASTRA Toolbox CUDA kernels using CuPy. KernelKit is written for user-customizable kernels, projectors and algorithms, and enables rapid prototyping of data-driven algorithms using the Python

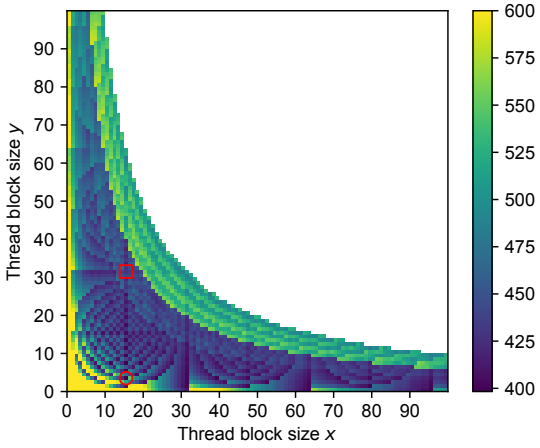


Figure 4.4: Conebeam backprojection time in microseconds for different CUDA thread block sizes on an NVIDIA GA102GL [RTX A6000] for a 2000-by-2000 slice and 32 projection angles of a 2000-by-2000 detector. The ASTRA Toolbox default thread block sizes, $(16, 32)$, are denoted by the red square. The optimum $(16, 4)$ is denoted by the red circle. The graphic is restricted to block sizes smaller than 100 for the purpose of visualization.

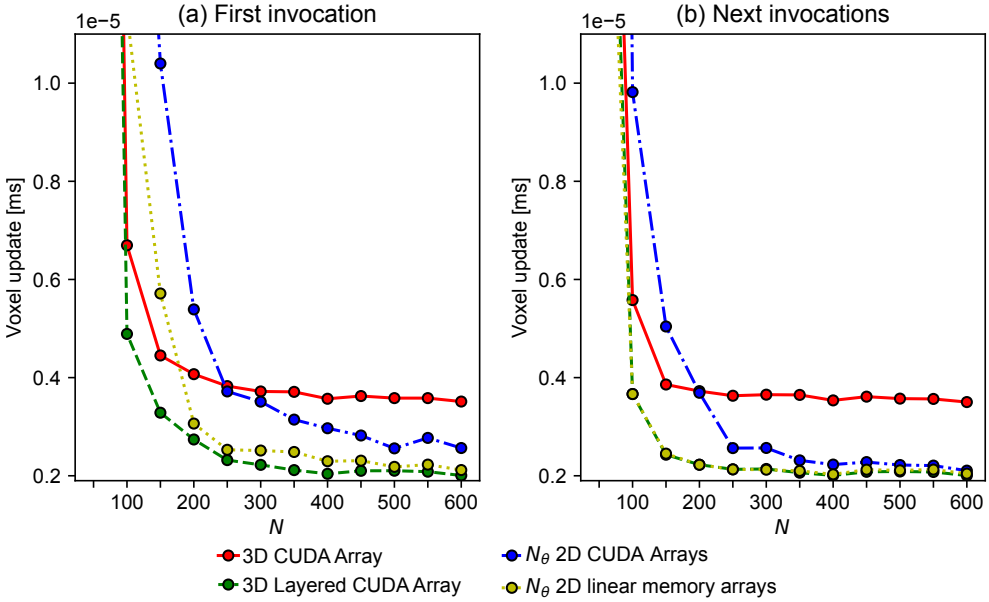


Figure 4.5: Time per *voxel update* of $\mathbf{A}^T$ on an (N, N, N)-sized volume using an NVIDIA GA102GL [RTX A6000]. (a) includes texture initialization on the first call to $\mathbf{A}^T$. (b) updates existing textures in any subsequent calls with new data of the same dimensions. Projections are $N_\theta := N$ angles of an (N, N) detector. A voxel update time is the backprojection time divided by $N_x N_y N_z N_\theta$.

ecosystem and philosophy. We envision it to serve as a minimalist back-end for high-level frameworks, as a package to develop projectors, and as a tool for high-performance applications that benefit from tuned algorithms. In this work we have focused on the voxel-driven conebeam backprojector, and demonstrated through patch-based learning, a tailored real-time algorithm, and runtime kernel compilation, that the Python ecosystem the software can now be used to implement efficient tomographic algorithms in several real-world use cases. In future work, we aim to extend the framework with new algorithm- and geometry-specific projectors [7], such as a matched forward- and backprojector, or kernels for curved detectors, for which the same principles apply. All in all, our package aims to accelerate the exploration and development of new high-performance and data-driven algorithms in CT.

Chapter 5

Scintillator Decorrelation for Self-supervised X-ray Radiograph Denoising

This chapter is based on the article: Adriaan Graas and Felix Lucka. "Scintillator decorrelation for self-supervised X-ray radiograph denoising". In: *Measurement Science and Technology* 36.6 (2025), p. 065415. DOI: 10.1088/1361-6501/addc06

5.1 Introduction

Imaging with X-rays is widely used for direct radiography and tomographic image reconstruction (CT, Computed Tomography). Thanks to its rapid acquisition and high spatial resolution, it is found in many applications of health care, industry (e.g., foreign object detection or non-destructive testing), and scientific research (e.g., experimental physics or biomedical sciences). An important topic within X-ray imaging is to mitigate noise from the radiographs, i.e., the raw radiation intensity measurements collected by X-ray detectors [138]. Noise does not only degrade the image quality, but also leads to streaks and other artifacts in the reconstructed images. The primary source of noise is *photon noise*, which is Poisson-distributed and caused by natural random variations in X-ray generation and attenuation [3, 4, 13, 15].

Current state-of-the-art methods for image denoising rely on deep learning, i.e., on training deep neural networks to map noisy input images to noise-free outputs [139, 140]. The most common training strategy is *supervised learning* [141], for which each noisy image from the training set is paired with a corresponding low-noise or noise-free *target* – the desired network output. For the situations that supervised targets are not available, *self-supervised learning* replaces them by noisy surrogates. For instance, in the strategy suggested by Noise2Noise [39], each surrogate target must be a second, statistically-independent, realization of the image. Since, in this case, the input provides no information that can help predicting the target's noise, an optimal network is only able to predict any noise-free image features that the input and target have in common. The requirement of two noisy images, however, is limiting for many imaging applications. This is especially the case for X-ray radiography and Computed Tomography, where auxiliary paired noisy images cannot be acquired with fast dynamics or low-dose scans.

In the recent years, research has focused on *unpaired* self-supervised approaches, i.e., strategies that do not need a secondary noisy realization. One particular promising category is that of *blind-spot networks* (BSNs). BSNs do not rely on a new image as surrogate target, but instead reuse a subset of pixels from the noisy image [41, 42]. These "blind spot" pixels, denoted by their indices J, need to be masked out in the input, denoted with u, for example via replacement by zeros or random values. Similar to Noise2Noise, the input provides no information for predicting the target's noise when u_{J^c} and u_J are statistically independent, which is satisfied when noise is pixelwise independent. On the other hand, spatially-extended image features in u_{J^c} provide information for predicting the signal of u_J, which a neural network can learn thanks to repeated occurrences of image features in the data. During inference, the full noise-free images can be recovered via execution of the BSN for all sets of blind spots.

BSNs are promising methods for denoising radiographs: They do not require modified acquisition, and are able to take advantage of large sets of noisy experimental data. Their main challenge for X-ray applications, however, is that detector instruments introduce local spatial correlations via blur. As a consequence, blind-spot networks propagate noise from u_{J^c} to u_J, rendering them ineffective. Mitigating blur and improving spatial resolution has been

an important research focus to reduce the effects of, e.g., final focal spot size, scattering, and system vibrations [142, 143, 144]. However, only blurs that affect the noise, i.e., when photon fluctuations correlate between nearby detector pixels, hamper the blind-spot denoiser.

In this article, we specifically target the blur caused within scintillator detectors. We show that, when correlations are uniform over the radiograph, they can be approximately reverted using a direct deconvolution in the frequency domain, i.e., without requiring a Wiener filter or an iterative algorithm. Our approach uses a deconvolution kernel that is obtained via an empirically-estimated scintillator point-response function (PRF), which describes the photon avalanche on conversion of X-rays into visible light. We show that it is robust against modest levels of additive Gaussian noise, and that the deconvolved radiographs restore the denoising potential of BSNs. In *Results I*, we verify this workflow using numerical simulations, an experimental phantom measured with Teledyne DALSA Xineos-3131 Caesium-Iodine scintillator detectors, and training of the blind-spot denoising method called Noise2Self [41]. In *Results II*, the workflow is applied to a large real-world experimental data set that is acquired for sparse-view dynamic X-ray tomography of fluidized beds. We provide an introduction in tomography in section 5.2.1, and the reader is referred to the textbooks [4, 145] for a treatment on this topic. To motivate denoising of radiographs as a preprocessing technique before reconstruction, section 5.5.1 compares Noise2Self with the Noise2Inverse denoiser in reconstruction space [40].

The paper is organised as follows: In *Background* (section 5.2) we review topics on X-ray imaging, computed tomography, deep learning for denoising, and blind-spot denoising with correlated noise. In *Method* (section 5.3) we discuss the noise model, the direct deconvolution, and neural network training. In *Results I* (section 5.4) we study the method on numerical and experimental data. In *Results II* (section 5.5) we investigate sparse-view tomographic image reconstruction on synthetic data and show results for experimental data from an ultra-sparse-angle X-ray set-up for imaging gas-solids fluidized beds.

5.2 Background

5.2.1 Tomographic reconstruction

The goal of tomographic reconstruction is to recover a 3D image $x \in \mathbb{R}^{N_x N_y N_z}$, corresponding to the discretization of $\mu(\eta)$ on a spatially uniform grid, from a set of projection images $[y_1, \ldots, y_{N_\psi}]$ acquired at N_ψ different positions of the X-ray source and detector, e.g., under different angles of an orbital or helical trajectory around the object. The discretization of all linear equations equation (1.4) is described by the operator $A\colon x \mapsto [y_1, \ldots, y_{N_\psi}]$ called the *forward projector* (cf. chapter 9 in [4]).

Tomographic reconstruction amounts to solving the ill-conditioned and often under-determined linear system $[y_1, \ldots, y_{N_\psi}] = Ax$ in an approximate way. A wide range of methods has been developed towards this purpose, see, e.g., the textbooks [3, 4], for an in-depth treatment of two reconstruction methods that will be used in our results. The first, the *filtered-backprojection* (FBP), is a two-step reconstruction technique that first applies a

suitable filter g to the projections and subsequently executes a *backprojection* operator A^{T}, i.e., a computational procedure approximating the transpose of A:

$$x^{\mathrm{FBP}} := A^{\mathsf{T}}[g \circledast y_1, \ldots, g \circledast y_{N_\psi}], \tag{5.1}$$

where $\circledast$ denotes convolution. FBP is similar to the Feldkamp-Davis-Kress (FDK) algorithm, which is tailored to a cone beam geometry instead of a parallel beam geometry. For our experiments we selected the Ram-Lak filter, cf. chapter 6 in [4]. The second, the *simultaneous iterative reconstruction technique* (SIRT), is an example of an algebraic iterative reconstruction technique (cf. chapter 11 in [4]). In this case, x^{SIRT} is an approximate solution to the constrained weighted least-squares problem

$$\underset{x \in \mathcal{C}}{\arg\min} \ \left\| Ax - [y_1, \ldots, y_{N_\psi}] \right\|_{M_1}^2, \tag{5.2}$$

found by means of a gradient-decent-type iteration

$$x^{(k+1)} = x^{(k)} + D_1 A^{\mathsf{T}} M_1 \left([y_1, \ldots, y_{N_\psi}] - A x^{(k)} \right) \tag{5.3}$$

on equation equation (5.2) for a fixed number of iterations K, i.e., $x^{\mathrm{SIRT}} := x^{(K)}$. D_1 and M_1 are diagonal matrices that contain the row and column sums of A, and $\mathcal{C}$ is the constrained solution space.

In section 5.5, FBP and SIRT algorithms are implemented using the ASTRA Toolbox software package [80]. On a modern graphics processing unit (GPU), FBP takes about a second of runtime, whereas SIRT can take several minutes, depending on the data dimensions.

5.2.2 Supervised and self-supervised denoising

Equation (1.3) in Chapter 1 is a simplified model of an ideal X-ray radiograph. It does not account for noise and does not describe all the physical phenomena that contribute to the factual measurement data (called *model mismatch*). The noise component mainly consists of *photon noise*, which stems from the quantum nature of light. When modeled with a Poisson probability distribution, it includes X-ray photon generation in the X-ray source, interactions with atoms in the sample, and detection in the scintillator (section 5.3.1) [3, 13, 14, 15]. Another large component of noise and model mismatch is due to *scattering*. X-ray photons that scatter in a direction away from the detector are indistinguishable from attenuated photons, and therefore accounted for by Beer-Lambert's law. However, photons that arrive at the detector via different paths than l_i in (1.3) of Chapter 1 (e.g., multiple scattering, multi-source set-ups), contribute to a model mismatch. Next to photon noise, a typically smaller noise component is due to electronics within the detector instrument. Their noise characteristics depend, e.g., on the the type of photodetectors used. Examples are charge-diffusion in CMOS (complementary metal-oxide-semiconductor) pixels, defective pixels, dark currents, amplifier noise, and digitization [3, 146].

Mitigating the effects of noise in X-ray measurements and CT reconstructions has been an important research focus for already many years [138]. Before the advent of machine learning algorithms, classical denoising methods such as BM3D, non-local means (NLM), or total variation (TV) had provided state-of-the-art results for images with photon noise or mixed Poisson-Gaussian noise. Most current research focuses on deep learned denoisers, and in particular convolutional neural networks (CNNs) [139, 140, 147]. We will describe a deep learned denoiser by a function f_θ that maps noisy images into clean images. Here θ denotes the set of learnable network parameters. During *training*, θ is optimized based on examples in a data set. The hope is that by leveraging statistical properties of the training data, the trained f_θ also performs well on unseen noisy images (*generalization*). In *supervised* training, f_θ is trained on pairs of noisy and clean images $(u, \hat{u}) \sim \mathcal{D}$ from a data distribution $\mathcal{D}$. The objective is then to solve

$$\arg\min_\theta \; \mathbb{E}_{(u,\hat{u})\sim\mathcal{D}} \, \|f_\theta(u) - \hat{u}\|_2^2 \,, \tag{5.4}$$

here given using the mean-squared error (MSE), the most common loss function. We write the loss as the minimization of an expected value, taken over image pairs $(u, \hat{u}) \sim \mathcal{D}$. Practically, however, data sets of example images consist only of a finite number of image pairs, and the expectation is replaced by the empirical mean (i.e., the average over a limited number of samples from $\mathcal{D}$). For brevity, we denote $\mathbb{E} \equiv \mathbb{E}_{(u,\hat{u})\sim\mathcal{D}}$ in the forthcoming sections, while in experiments we will evaluate it with the empirical mean.

For many applications, obtaining the clean images $\hat{u}$ needed for supervised learning is impractical or impossible. *Self-supervised* learning therefore tries to modify the objective in equation (5.4) in such a way that $\hat{u}$ is not needed anymore [14]. The Noise2Noise principle [39], discussed in the introduction, does this by replacing $\hat{u}$ with a second noisy image, assuming the images are statistically independent. While its principle was used in, e.g., electron microscopy [148, 149], paired acquisition often poses challenges for real-world applications, and several methods have therefore been suggested to construct pairs from single noisy images. One example is Neighbor2Neighbor, which constructs image pairs via subsampling [150]. Other alternatives are *zero-shot methods*, such as the Deep Image Prior [147] or Zero-Shot Noise2Noise [151]. Zero-shot learning, however, is typically slower and less accurate than CNNs trained on a data set.

5.2.3 Related work

When noise is correlated, $\mathcal{J}$-invariance can in principle still be used by letting the $J \in \mathcal{J}$ in definition 1.8.1 consist of masking regions larger than pixelwise grids. MM-BSN (Multi-Mask BSN), for example, applies differently-shaped masks to suppress noise with non-uniform correlation structures [152]. Indeed, when J covers larger correlated regions, the property $f_\theta(u) - \hat{u} \perp u - \hat{u}$ may hold again, therefore removing the cross-term in equation (1.14). However, schemes that require extended masking generally cause blurring of the estimate, since information about sharp image features is generally contained in a close neighborhood of the target pixels.

Other approaches in the literature mitigate correlated noise by using pixelwise subsam-

pling in conjunction with BSNs. Example architectures are AP-BSN [153] and SS-BSN [154]. In AP-BSN, the input is subsampled to s^2 small images using a pixel-shuffle downsampling operation. With $s \in \mathbb{N}^+$ denoting the noise correlation radius, the noise in each subsampled low-resolution image becomes uncorrelated. After denoising the subimages with a BSN, the results are upscaled to a high-resolution output using a learned refinement to overcome aliasing. It has been observed that AP-BSN can be difficult to apply to scientific images [149]. Noise2SR [155] and M-Denoiser [149] extend the subsampling method by randomization and shattering, rather than pixel-shuffling.

5.3 Method

Instead of masking large regions, or employing subsampling, our method obtains $\mathcal{J}$-invariance by reverting the process that causes the spatial correlation. Section 5.3.1 first discusses the origin of correlation in X-ray radiographs obtained from scintillator detectors. Section 5.3.2 then motivates the use of a direct deconvolution, and section 5.3.3 and section 5.3.4 outline a correlation kernel estimation and neural network training procedure.

5.3.1 Scintillator noise model

Scintillator detectors have a widespread use, e.g., in medical devices and scientific laboratories including synchrotron light sources [142]. In these detectors, a crystal converts incident X-rays into visible light from the optical spectrum, which can subsequently be detected with an array of photodetectors such as CMOS active pixels [4, 156]. These pixels convert the light to electric charge, which is stored until it is read out by the detector electronics. For the X-ray wavelengths produced by cone beam sources, common choices for crystals, also called *phosphor screens*, are Caesium-Iodine (CsI) and Gadolinium Oxysulphide (GadOx). These crystals consist of high atomic-number elements that increase the probability of photoelectric interaction.

The scintillation event is responsible of the spatial correlation found in X-ray radiographs: The one-to-many X-ray conversion gives rise to a *photon shower* consisting of hundreds to thousands of optical photons. The optical photons are emitted in isotropic direction, and therefore spread out over a range of photodetectors at the scintillator exit plane. The shape of the resulting blur is quantified by a point-response function (PRF), defined as an image after illumination of the detector with an infinitely narrow X-ray pencil beam [157]. This is also commonly termed a PSF (point-spread function). The spread of the blur mainly depends on the thickness of the phosphor screen [158], but the specific blur shape depends on manufacturing details (e.g., the use of Thallium (TI) doping, transmission of charge through an amorphous silicon, or needle-grown structure of CsI). Manufacturers offer detectors with phosphor screens in a variety of thicknesses, as applications require different trade-offs between detection efficiency and spatial resolution. An analytical model of the blur can be approximated via Monte Carlo simulation [157].

When the PRF is assumed to be spatially-invariant over the radiograph, it can be modeled by a uniform convolution with a kernel h. We replace the notation for the idealized

radiograph I obtained from Beer-Lambert's law, equation equation (1.3) in Chapter 1, with the following three equations:

$$\begin{aligned} \lambda_i &:= I_i^0 \exp(-\textstyle\int_{l_i} \mu(\eta)\,\mathrm{d}\eta) && \text{(noise-free)} \\ \hat{I} &\sim \text{Poisson}\,(\lambda) && \text{(independent noise)} \\ I &:= h \circledast \hat{I} && \text{(correlated noise).} \end{aligned} \tag{5.5}$$

Here, λ is the noise-free signal, $\hat{I}$ is a multivariate random variable describing a radiograph with pixelwise-independent photon noise, and the radiograph I is now redefined via the discrete convolution of independent Poisson variables. At high photon counts, I approximates a multivariate normal distribution, i.e.,

$$I \sim \mathcal{N}(H\lambda, H\Lambda H^{\mathsf{T}}), \tag{5.6}$$

with H being the linear operator associated with h, and $\Lambda := \text{diag}(\lambda_1, \ldots, \lambda_{N_u N_v})$ a diagonal matrix. Note that the covariance matrix, $H\Lambda H^{\mathsf{T}}$, reflects the signal-dependent nature of Poisson noise.

The convolutional model equation (5.5) is a linear and uniform approximation of the detector response, allowing efficient deconvolution in tomographic pipelines (section 5.5.2). However, under certain specific imaging conditions, one or multiple of the following sources of model mismatch may become non-negligible: (i) The non-uniform nature of the PRF, due to its dependence on the incident X-ray angle and energy spectrum (the latter is known as Swank noise [159, 160]); (ii) Detector electronics, which can cause correlation through charge-sharing [161] or regional thermal effects in the photodetectors. While oftentimes negligible [146, 156], their effect is known to be more prominent at low photon counts; (iii) Scintillator responses become non-linear for high photon rates [3], and some detector instruments use heterogeneous pixel technology. In the forthcoming sections, we will investigate the correlation structures of real-world radiographs, and simulate anisotropic Gaussian noise, which is the commonly-assumed model for electronic noise.

5.3.2 Direct deconvolution

The correlation structure due to the PRF, as described in equation equation (5.5), cannot be addressed sufficiently by the blind-spot denoising strategies discussed in section 5.2.3. Convolution kernels, even with small radii, require large masking schemes or high subsampling factors, as the radius of correlation is twice the radius of convolution. This motivates our approach using *direct deconvolution*, which is available thanks to the noise model. Direct deconvolution is an approximate inverse to convolution, and an established technique in X-ray radiography for filtering noise and improving sharpness [162]. Using the Fourier transform $\mathcal{F}$, it is computed as

$$H^+(I) := \mathcal{F}^{-1}\left[\mathcal{F}[I] \cdot \frac{\mathcal{F}[h]^{\mathsf{H}}}{|\mathcal{F}[h]|^2}\right], \tag{5.7}$$

with $(\cdot)^{\mathsf{H}}$ denoting the conjugate transpose.

Equation (5.7) is efficient because convolution and deconvolution become element-wise multiplications in the frequency domain, and the Fourier transform can be computed quickly using the Fast Fourier Transform (FFT). When applied as a preprocessing step prior to a convolutional neural network, the cost of the deconvolution is typically negligible: CNNs contain already many convolutional blocks, each comprising multiple filters and working on multi-channel inputs.

Poisson case We first illustrate that, in the case of pure Poisson noise, deconvolution immediately enables blind-spot denoising. Recall that Noise2Self is based on the idea that the cross-term in the optimization objective, equation equation (1.14), should vanish. If we were to use the ordinary input, i.e., $u := I$ and $\hat{u} := H\lambda$,

$$\mathbb{E}\,\langle f_\theta(I) - H\lambda, I - H\lambda\rangle \tag{5.8}$$

would not be zero, as I_{J^c} is not statistically independent of I_J (cf. the covariance in equation equation (5.6)). However, preprocessing the image with a deconvolution gives $u := H^+ I \approx \hat{I}$. The covariance of u is then:

$$\mathbb{E}\,[u_i u_j] = (H^+ H \Lambda H^{\mathsf{T}} H^{+\mathsf{T}})_{ij} \approx \Lambda_{ij}, \tag{5.9}$$

which is a diagonal matrix, thus does not contain cross-correlations. In the theoretical case, with pure Poisson noise and a known PRF, the deconvolved images $u := H^+ I$ therefore restore the potential for blind-spot denoising.

Poisson-Gaussian case We now extend the model to real-world noise. Compared to the Poisson case, real-world data is further degraded with additive noise, which is known to produce high-oscillatory "checkerboard artifacts" after deconvolution. To see this, consider an additive anisotropic Gaussian noise $\varepsilon_i \sim \mathcal{N}(0, \sigma_i^2)$. By the linearity of deconvolution, we have that $H^+(I + \varepsilon) \approx \hat{I} + H^+\varepsilon$, showing that deconvolution recovers the pixelwise independent noisy image $\hat{I}$ at the cost of adding a checkerboard component $H^+\varepsilon$. Since the the checkerboard artifacts are spatially-structured, it adds a pattern that can be reproduced by blind-spot denoisers.

Instead of deconvolving with h, we will look for a $\tilde{h}$ that minimizes the overall spatial correlation better by balancing the two types of noise. Let $\widetilde{H}^+$ denote the associated deconvolution operator, and $u := \widetilde{H}^+(I + \varepsilon)$ the deconvolved radiograph. Similar to the Poisson

It is worth noting that $\hat{I}$ can be blurred more extensively than what is expected only from the scintillator PRF, due to, e.g., the finite focal spot aperture in the source, or instrument vibration [3, 142, 163]. Several methods exist for estimation and resolving such blurs, e.g., with a procedure involving a sharp-edge phantom [163]. However, using methods based on sharpness criteria for our purpose, which is decorrelation, can cause a model mismatch and lead to, e.g., ringing artifacts in iterative schemes. It is therefore that we will use a statistically-estimated PRF with direct deconvolution, and not the more sophisticated iterative Poisson-deconvolution algorithms such as Richardson-Lucy.

case equation (5.8), the goal is to maximize the statistical independence of the blind spot with the non-masked pixels. For a blind-spot u_i and a pixel u_j in the PRF's support, the covariance is

$$\mathbb{E}\left[u_i u_j\right] = (\widetilde{H}^{+}(H\Lambda H^{\mathsf{T}} + \mathrm{diag}(\sigma_1^2, \ldots, \sigma_{N_u N_v}^2))\widetilde{H}^{+\mathsf{T}})_{ij}. \tag{5.10}$$

Since the inner term is the covariance matrix of $I + \varepsilon$, we can see that $\widetilde{H}^{+}$ should perform a diagonalization of the covariance matrix. In the case of a uniform signal λ_{const} and noise variance $\sigma^2_{\mathrm{const}}$, the closest solution to diagonalization would be given by the inverse of

$$\widetilde{H} = \left(HH^{\mathsf{T}} + \sigma^2_{\mathrm{const}}/\lambda_{\mathrm{const}}\mathrm{Id}\right)^{\frac{1}{2}}, \tag{5.11}$$

up to a factor of scaling (see Appendix 5.8 for the procedure and additional remarks). This shows that the additive noise has a regularizing effect. Note that $\widetilde{H} \to H$ as $\sigma^2_{\mathrm{const}} \to 0$, meaning that in the limit of smaller additive noise we return to the Poisson case, for which the scintillator PRF is optimal. In general, however, the solution to equation (5.10) can not be expressed analytically anymore.

5

5.3.3 PRF estimation

Algorithm 2 outlines a data-driven estimation procedure for the convolution kernel associated with $\widetilde{H}$. It relies on the availability of a set of radiographs $\{I^{(1)}, \ldots, I^{(T)}\}$ between which only the noise changes (for example, the radiographs acquired from a static object with a stationary source-detector set-up). It estimates a sample covariance matrix of the noise, and then extracts its latent convolution kernel. The presentation here excludes implementation details, such as how to sample at the image boundaries, or how to centralize and normalize the kernel. These can be found in the algorithm code in Appendix 5.7, and further details are given in Appendix 5.8.

The algorithm takes as input the radiographs, a temporal stride Δt, and a set of indices $\mathcal{J}_{\mathrm{corr}}$. Line 1 first removes the background signal via subtraction of $I^{(t+\Delta t)}$ from $I^{(t)}$. Compared to subtracting the mean, using radiograph differences can be more robust to slow source current fluctuation during an X-ray scan. Δt is a parameter that should be chosen such that noise in the frames is temporally uncorrelated. Line 2 transforms the noise such that it has uniform variance, and line 3 computes correlation coefficients for the pixels i and $i + j$, where $j \in \mathcal{J}_{\mathrm{corr}}$. Here, the index set $\mathcal{J}_{\mathrm{corr}}$, for instance a rectangular neighborhood, must be large enough to include all pixels within the correlation range of the pixel i. As the correlation distance is typically unknown, a user should increase $\mathcal{J}_{\mathrm{corr}}$ until sampling with a larger radius has no significant effect anymore on the algorithm outcome. After forming the sample covariance matrix, line 5 retrieves $\tilde{h}$ using the matrix square root [164].

For the data set of radiographs, we recommend a static object with similar experimental conditions as the denoising data set. When such data is unavailable, a static image region extracted from multiple frames of an arbitrary acquisition may already give good results. In all cases, it is necessary to avoid detector regions where the correlation is affected, e.g., via clipping of intensity values to the detection limit.

Algorithm 2:
In: Radiographs $\{I^{(1)}, \ldots, I^{(T)}\}$, temporal stride Δt, indices $\mathcal{J}_{\text{corr}} \subset \mathbb{Z}$
Out: Convolution kernel $\tilde{h}$

1: Subtracting the background signal gives the *noise differences*:

$$D^{(t)} \leftarrow I^{(t)} - I^{(t+\Delta t)}$$

2: Stabilize D using the sample variance for all pixels i:

$$M_i^{(t)} \leftarrow \frac{D_i^{(t)}}{\sqrt{s_{ii}^2}} \text{ with } s_{ii}^2 \leftarrow \frac{1}{T}\sum_{t=1}^{T} D_i^{(t)} D_i^{(t)}$$

3: Estimate correlation coefficients $\tilde{c}_j$ for all $j \in \mathcal{J}_{\text{corr}}$:

$$\tilde{c}_j \leftarrow \frac{1}{N_u N_v T} \sum_{i=1}^{N_u N_v} \sum_{t=1}^{T} M_i^{(t)} M_{i+j}^{(t)}$$

4: Construct a small Toeplitz sample covariance matrix $\widetilde{C}$ by setting the diagonals to $\tilde{c}_j$.
5: Compute the matrix square root of the positive semi-definite $\widetilde{C}$:

$$\widetilde{H} \leftarrow \sqrt{\widetilde{C}}$$

The kernel $\tilde{h}$ can be retrieved as the first row or column of $\widetilde{H}$.

5.3.4 Neural network

Deconvolved images can now be used as training data for blind-spot networks. For our experiments we will optimize the loss

$$\arg\min_{\theta} \; \mathbb{E} \; \|\log f_\theta(u) - \log u\|_2^2 \,, \tag{5.12}$$

with $u := \widetilde{H}^{+} I$. The L2-norm is well-known to overestimate high signal in radiographs due to heteroscedasticity of the noise. Several solutions are used in practice, such as a Poisson loss or Anscombe transform [165]. In the high photon count regime, an alternative is to take the log-transform, which achieves a loss that linearly relates to the reconstruction that we want to compute from the denoised data (see equation (1.4)).

For the network architecture f_θ of (5.12), we will employ a standard U-Net [38] in all experiments of the result sections. This is an image-to-image architecture with a symmetrical encoder and decoder, using convolutional downsampling operators and bilinear upsampling operators, respectively. In our implementation, the network consists of 256 feature maps in the first level, and uses skip connections between the encoder and decoder. We note that Noise2Self and Noise2Void can also be used in conjunction with other image-to-image architectures, such as DnCNN [166].

To make the U-Net $\mathcal{J}$-invariant, we make use of the default masking procedure of Noise2Self, which entails a zero-value replacement of pixels on a 3-by-3 grid. During in-

ference, the network is executed on all grids, and the resulting outputs are assembled into a single image.

5.4 Results I: Radiograph denoising

Before demonstrating our denoising approach on experimental data from a dynamic experiment without ground truths, we first validate each component in the pipeline of kernel estimation, direct deconvolution, and blind-spot denoising.

Correlation maps A useful quantity to visualize spatial correlation is the summed correlation of each individual pixel with its neighborhood. This quantity can only be computed with a sufficiently large set of static radiographs $\{I^{(1)}, \dots, I^{(T)}\}$. Denote the *correlation map* with $m \in \mathbb{R}^{N_u N_v}$. For a pixel $m_i(I)$ in the map,

$$m_i(I) = \sum_{j \neq i} \frac{\mathrm{Cov}(I_i, I_j)}{\sqrt{\mathrm{Var}(I_i)}\sqrt{\mathrm{Var}(I_j)}} \approx \frac{1}{T} \sum_{j \neq i} \sum_{t=1}^{T} M_i^{(t)} M_j^{(t)}. \tag{5.13}$$

Here, the first equality sums the Pearson correlation coefficients of pixels (I_i, I_j) and the second equality reuses the notation for variance-stabilized noise from algorithm 2.

5.4.1 Deconvolution of simulated radiographs

In this section, we will use radiographs that are simulated from a numerical foam [167], i.e., a numerical phantom that consists of a single-material cylinder with randomly-positioned spherical voids. This phantom will also be used in Results II (see figure 5.6 for a horizontal slice through the foam). Here, it is configured to contain 100 spheres per unit of cylinder, a low-dose current of $I^0 := 1000$, and a parallel beam geometry. We use the phantom software [167] to calculate the ground truth radiograph, see λ in (5.5), analytically, and then generate 5,000 128-by-128 noisy radiographs by adding simulated Poisson noise and additive Gaussian noise. This simulation yields repeated static radiographs, making them suitable for PRF estimation and visualization of local correlations with (5.13). We first test the accuracy of PRF estimation (algorithm 2), and then inspect the residual correlations after deconvolution with Gaussian noise.

From algorithm 2, errors are expected due to variance stabilization (line 2), a finite number of statistical samples (line 3), and the matrix solver (line 5). To convey an impression, the first row of figure 5.1 applies the algorithm on the 5,000 radiographs. The simulation only applies Poisson noise and blurs the result using a 5-by-5 Laplacian PRF (scale parameter $b = 0.5$). The accuracy of $\tilde{h}$, due to the variance stabilization (line 2 in algorithm 2), was 1.2% relative error after $T = 30$ static noisy images, and 1.1% after $T = 200$. Further improvement was not observed, due to nonuniform regions in the radiograph (see Appendix 5.8). The error due to the matrix square root was found to be negligibly small for the Laplacian kernel.

The three rows of figure 5.1 display three simple noise conditions for the simulated radiographs, namely: Pure Poisson noise, isotropic Gaussian noise, and anisotropic Gaussian

noise. In the pure Poisson case, the difference $\widetilde{H}^{+}I - \hat{I} \approx 0$ confirms that deconvolution with an estimated PRF still reconstructs the photon noise with very high accuracy. The histograms in the residual correlation plot furthermore confirm that deconvolution removes the correlations.

The isotropic and anisotropic cases of figure 5.1 illustrate two problems that can occur at the extremes of additive pixelwise independent Gaussian noise ε. At high levels of isotropic noise (middle row), correlations can become signal-dependent, i.e., the spatial distribution of high-intensity regions in $m(\widetilde{H}^{+}I)$ follows the radiograph intensities. Strongly anisotropic additive noise (bottom row) cannot be accounted for by a uniform kernel. When this noise is added to the lower image part, deconvolution partially decorrelates the photon noise in the upper image, while reintroducing negative-correlation checkerboard artifacts in the lower image. The positive correlation and negative correlation are visible as distinct peaks in the histogram.

The numerical experiments provide practical insights into PRF estimation. In scenarios with low photon counts or strongly anisotropic noise, uniform deconvolution is unable to fully eliminate the correlations. In the next section, we will therefore first examine the correlation maps of real-world radiographs. In the subsequent section 5.4.3, we explore how residual correlations after deconvolution are handled by blind-spot denoisers.

5.4.2 Deconvolution of an experimental phantom

In this section and section 5.4.3, the data consists of 1,000 repeated radiographs from two static "bubble phantoms". These radiographs show a PMMA (polymethyl methacrylate) cylinder that is filled with granular particles and two 23 millimeter polystyrene spheres. The set-up is described in detail in [168]: It consists of three fixed cone beam sources directed at three Xineos-3131 CsI scintillator detectors with CMOS pixel technology. The detectors are oriented as portraits and operate in a 1548-by-550 virtual region of interest to improve the read-out speed.

Using algorithm 2 with $T = 200$, $\Delta t = 3$ and an 11-by-11 correlation region $\mathcal{J}_{\text{corr}}$, we retrieve three kernels for the Xineos-3131 detectors. Since the differences between the kernels were found to be less than 0.5%, we limit the results to the first detector. Figure 5.2 shows a crop of the first detector radiograph, imaged with a tube voltage of 120 kVp and anode current of 1.5 mA. Using the 1,000 radiographs, the correlation map (equation equation (5.13)) before and after deconvolution is inspected. Before deconvolution, correlation is highly uniform, and an average pixel correlates about 356% with its surroundings, i.e., the mean of the black histogram. After deconvolution, the distribution is centered.

An important practical result is the uniform background of the correlation maps in figure 5.2. This indicates that, for this set-up, spatial correlation is not strongly dependent on the signal (i.e., the phantoms, cylinder or metal components are not visible in the correlation maps). Following the results of the previous section, we therefore expect the PRF to be approximately uniform, and the additive noise to be neither strong nor very anisotropic.

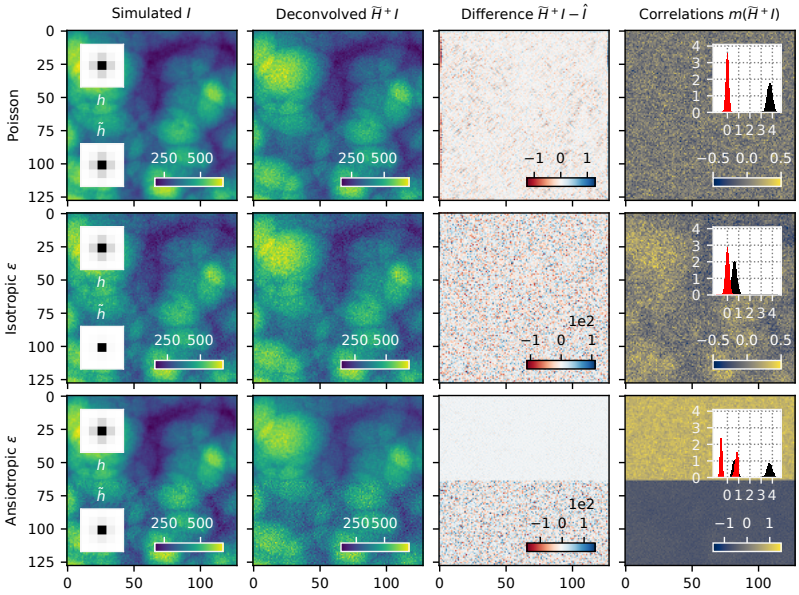


Figure 5.1: Kernel estimation and deconvolution evaluation: Simulated projection data of spheres impacted by high-noise (section 5.4.1). The **top row** contains pure Poisson noise. The **middle row** adds isotropic Gaussian noise with $\sigma^2 = 400$. The **bottom row** only adds the Gaussian noise in the lower half-plane. The insets on the residual correlation maps (right column, see equation (5.13)) show histograms of the summed correlation over the whole image before (black) and after (red) deconvolution.

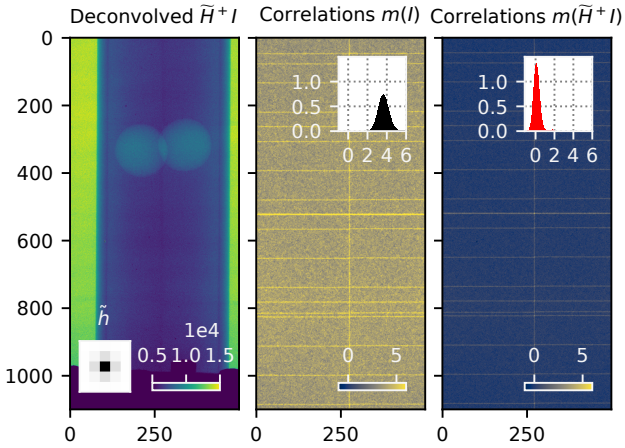


Figure 5.2: Deconvolution of 1,000 radiographs of a static PMMA cylinder containing two polystyrene balls, metal equipment, and non-uniform background radiation (cropped view, section 5.4.2). **Left** displays one radiograph and the kernel $\tilde{h}$. The **middle and right** show the correlation maps of I and $\widetilde{H}^+ I$. The lines of high correlation are due to inpainting of defective pixels during preprocessing. The probability density histograms show shifting and narrowing of the correlation distribution (black→red).

5.4.3 Blind-spot denoising with Noise2Self

We test the suitability of the deconvolved phantoms for blind-spot denoising. First, the U-Net is trained on the deconvolved radiographs $\{\widetilde{H}^+ I^{(1)}, \ldots, \widetilde{H}^+ I^{(1000)}\}$ of figure 5.2. As a ground truth to evaluate the denoising task, we use the mean (cf. equation (5.6)) over the deconvolved radiographs, i.e.,

$$\widetilde{H}^+ H\lambda \approx \frac{1}{T}\sum_{t=1}^{T} \widetilde{H}^+ I^{(t)}. \tag{5.14}$$

While training with static radiographs is not yet representative of a real-world data set, the experiment allows a better investigation of the network response to correlation structures.

Figure 5.3 shows a zoom-in on the noisy image, ground truth image, and two denoising results at different points during the training stage. The results show that the deconvolved data is denoised well and that the network even recovers some of the stacked granular particles that are difficult to discern in the raw radiograph. The ground truth equation (5.14) is not fully recovered by N2S: As we will see, this is not due to the remaining correlations, but is the usual performance of the denoiser. Noise overfitting occurs after extensive training, leading to artifacts in the image (cf. bottom right plot in figure 5.3). This is especially the case for the inpainted dead-pixel lines, which lead to a correlation structure that is more easily learned by the network (cf. figure 5.2).

To further validate our approach, we compare the resulting network against two baselines; f_{raw} using the original, i.e., correlated, noisy data, and f_{syn} using synthetic, uncorrelated, noisy data. Both are trained in exactly the same way as f. The synthetic data is generated by adding Gaussian noise to the ground truth using the sample variance extracted from the real data, and is therefore representative of a correlation-free baseline. Figure 5.4 compares the three networks and displays their evaluation on a single radiograph. For f_{raw}, the output remains noisy and displays checkerboard artifacts (see the figure inset). The denoising results, $\log f(\widetilde{H}^+ I)$, are very similar to the synthetic data, $\log f_{\text{syn}}(\widetilde{H}^+ H\lambda + \varepsilon_{\text{syn}})$. Both f and f_{syn} achieve similar denoising performance, although the synthetic network did not suffer from overfitting.

To see how the remaining correlation structure in the deconvolved data (i.e., the map $m(\widetilde{H}^+ I)$ in figure 5.2) affects f, we add different realizations of synthetic, uncorrelated noise ε to the same input $\hat{u}$ and calculate

$$p_i(f_\theta, \hat{u}) = \mathbb{E}_\epsilon \sum_{j \neq i} \frac{\operatorname{Cov}(f_\theta(\hat{u}+\epsilon)_i, (\hat{u}+\epsilon)_j)}{\sqrt{\operatorname{Var}(\epsilon_i)}\sqrt{\operatorname{Var}(\epsilon_j)}}, \tag{5.15}$$

which measures how much of the added noise is propagated from surrounding pixels to a target pixel. The result, the bottom row of figure 5.4, shows that f_{raw} relies everywhere on values from the local pixels, an indication that it learned to exploit correlations in the noise. On the other hand, f and f_{syn} only use the local neighborhoods where there are sharp image features (e.g., the dark spot and phantom edges have higher intensities in $p(f_\theta, \hat{u})$).

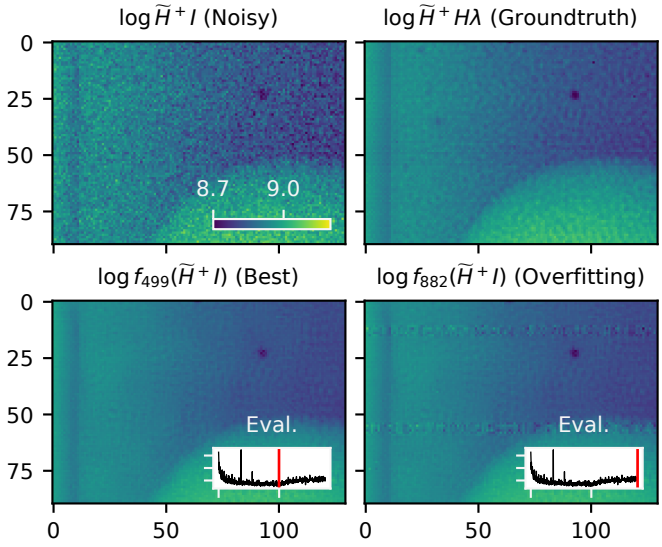


Figure 5.3: Training Noise2Self on the deconvolved data. **Top row**: Noisy and ground truth deconvolved radiographs (zoom into the upper-left part of figure 5.2). **Bottom row**: the trained Noise2Self U-Net evaluated on a single noisy image at an optimal point (epoch 499) and overfitting stage (epoch 882) during network training. *Eval.* plots the MSE evaluation of the trained network f_i at epoch i with the ground truth.

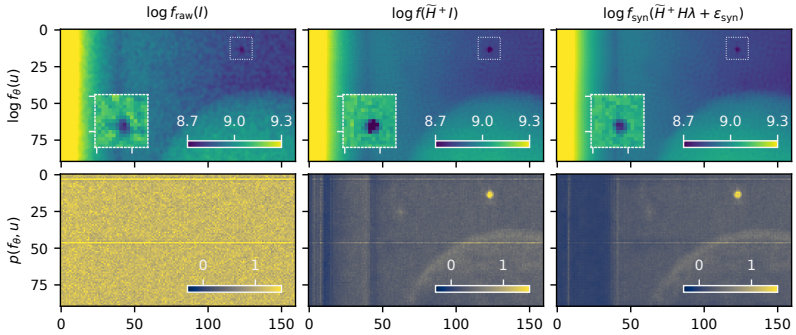


Figure 5.4: Blind-spot-denoised radiographs (figure 5.2), as well as their input-to-output correlation equation (5.15) summed over a 3-by-3 neighborhood (section 5.4.3). The **left plots** use unprocessed raw data, the **middle plots** deconvolved data, and the **right plots** synthetic independent noise. The granular background in the cylinder is a physical phenomenon due to stacking of particles. The output of f_{raw} displays checkerboard artifacts (inset in top left plot).

Moreover, both f and f_{syn} have the same average background values, an indication that f was not able to fit to remaining correlations in $m(I)$ in figure 5.2. Overall, the results in this section show that deconvolution can restore the blind-spot denoising performance of uncorrelated noise, but that care should be taken not to introduce easy-to-distinguish correlation structures such as inpainted pixels (figure 5.3).

5.5 Results II: Sparse-view Computed Tomography

For X-ray Computed Tomography, self-supervised denoising can be executed either in the projection domain [169, 170], e.g., via a Noise2Noise-like mapping between adjacent projections, or in the reconstruction domain, e.g., via reconstructions of angular subsets [40, 171] or via neighboring slices in a parallel-beam reconstruction [172]. The reconstruction domain has the advantage that the image features are compact and local, favoring denoising with CNNs, and often provides lower noise statistics due to averaging of noise from multiple projection angles [3, 4]. On the other hand, noise is spatially local in the projection domain, and can be more difficult to remove once propagated over the reconstruction domain by the CT algorithm. Self-supervised methods that work in both domains [173] are promising but are still too difficult to be applied for high-resolution volumetric tomographic data due to their high computational demands.

One promising use of BSNs is projection-domain denoising for *sparse-view CT*, i.e., reconstruction from an undersampled set of angular projections [174]. While dense samplings are typically required to recover high-resolution object details [4], sparse-view CT is advantageous when very short scanning times and/or very low radiation exposures to the sample are crucial. In section 5.5.1, we first compare self-supervised denoising in projection and reconstruction domain on simulated sparse-view CT data. Then, in section 5.5.2, we apply blind-spot denoising to real radiographs from dynamic ultra-sparse view CT using our PRF estimation and direct deconvolution pipeline.

5.5.1 Denoising of projections and CT images

In this experiment, we compare our denoising approach with the self-supervised denoiser in the reconstruction domain using different numbers of scan angles N_ψ and for two reconstruction methods, namely FBP and SIRT. While such comparison ultimately relies on the noise characteristics and capacities of the machine learning architecture, we will show that projection-domain denoising can become advantageous in the case of sparse-view geometries.

For projection-domain denoising, we will use Noise2Self, as before, and for reconstruction-domain denoising Noise2Inverse (N2I) [40]. In its simplest configuration, N2I splits projections into sets of even and odd indices, enabling even-to-odd denoising of filtered-backprojection (FBP) slices. This resembles Noise2Noise [39] on reconstructed images (section 5.1, section 5.2.2). During evaluation, even and odd network outputs are averaged.

The simulated radiographs are again of the parallel-beam numerical data used in section 5.4.1 [167], now with $N_u, N_v = 512, 128$ pixels. We use $I^0 = 10^4$ photons for generating the signal. However, as pure Poisson noise is too optimistic compared to the noise levels

in real-world data, we add 10^4 counts of zero-mean Poisson fluctuations. This follows a Gaussian distribution closely [3], and reflects a radiograph of which 50% of the signal is due to scatter. We do not model the scintillator blur, as this experiment aims to isolate the aspects of denoising in the two domains, and deconvolution can be performed with either method. The image features are therefore more complex in the projection domain (see figure 5.1) than in the reconstruction domain (figure 5.6). Both N2I and N2S are trained with the same U-Net configuration and stopped before noise fitting using a ground truth criterion.

Figure 5.5 displays the results after training and reconstruction on two reconstruction algorithms: FBP and SIRT (section 5.2.1). Focusing on FBP first, in (a), we see N2I performs remarkably well, as it even outperforms FBP reconstruction from clean projections. Figure 5.6 shows that this is possible because N2I smooths sparse-angle artifacts in image space. N2S, on the other hand, follows the lower limit of the shaded area, meaning it is close to the FBP solution with clean projections. For denser angles, its error (note the $\log_2$-scale) contributes significantly to the reconstruction. We observed that the high error of N2S is mostly due to the low number of training samples (the N_ψ projections) relative to the high image complexity.

In (b) of figure 5.5, N2I and N2S are combined with SIRT reconstruction using box constraints to restrict valid solutions to $[0, 1]$, cf. equation equation (5.3). The box constraints, which turn SIRT into a non-linear method, are chosen to illustrate that N2S can both be used with linear and non-linear methods, and will also be used in section 5.5.2. The algebraic reconstruction technique is known to be better-suited for sparse-angular geometries [4]. By our knowledge, N2I has not been demonstrated successfully with SIRT results before. Our results indeed show artifacts in the N2I-denoised images, suggesting that N2I may not be able to resolve the finer object details, possibly due to the complex noise structure after the iterative reconstruction. Blind-spot denoising with N2S, on the other hand, is independent of the reconstruction algorithm that follows. SIRT started with N2S-denoised data allows therefore for a more accurate reconstruction than N2I in the sparse-angle segment. Figure 5.6 shows that for the sparse-angle case with $N_\psi = 32$, the best denoising strategy is to combine N2S with SIRT.

5.5.2 Ultra-sparse CT of single-bubble-injected fluidized beds

We close the article by blind-spot denoising real-world projections of gas-solids fluidized beds and performing a subsequent SIRT reconstruction. A gas-solids fluidized bed is a mixture between a gas and a granulate material that behaves similarly to a fluid. Bubbles, i.e., the voids in the bed, are studied in laboratory-scale experiments, both via X-ray radiography and timeframe-by-timeframe dynamic CT. Typically, statistical quantities about the shapes and sizes of bubbles are inferred using image analysis techniques such as segmentation and tracking. In this particular example, we look at single-bubble-injected data: The injected gas is regulated at the set-up inlet such that only a single bubble travels through the granulate material at the time.

The ultra-sparse set-up at Delft University of Technology is specifically built to image

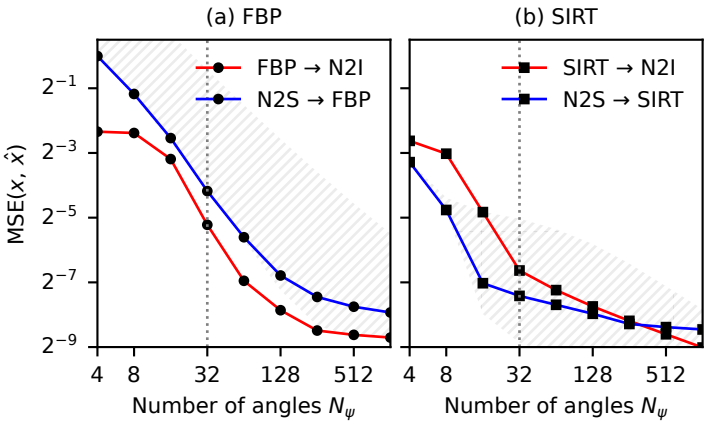


Figure 5.5: N2S and N2I combined with N_ψ sparse-view FBP and SIRT reconstructions (section 5.5.1) of the spherical phantoms [167], see figure 5.6 for a plot at the vertical indication line. Each data point is an individually trained U-Net [38] on N_ψ angles that is optimally stopped using ground truths. The limits of the shaded areas mark reconstruction from clean and noisy data.

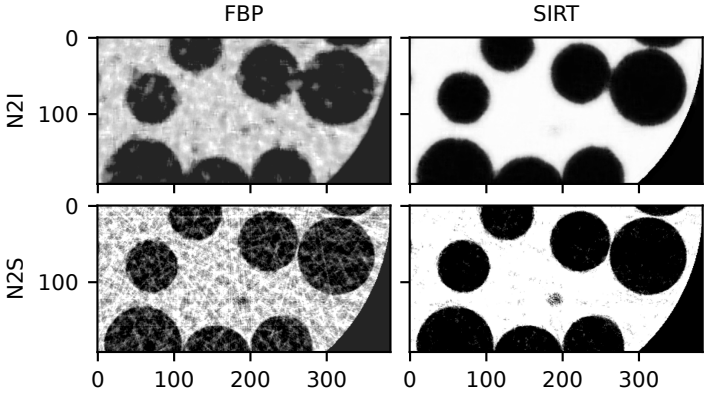


Figure 5.6: 192-by-384 region of interest taken from the $N_\psi = 32$ simulated reconstructions (see figure 5.5) at the central volume slice of the binary phantoms [167]. See figure 5.1 for an impression of the projections. A circular mask is applied to reduce the contribution of artifacts outside of the object in the MSE score of figure 5.5.

the high-velocity bubbles while they travel towards the bed's surface. It consists of three pairs of sources and Caesium-Iodine detectors (section 5.4.2), positioned into an equilateral triangle. The synchronized radiographs obtained from the set-up suffer extraordinarily from noise, on one hand due to their short exposure intervals used in the X-ray detectors, and on the other hand due to the cross-scatter of photons from the non-facing sources. This significantly deteriorates subsequent image reconstructions [168], since high levels of noise in iterative methods lead to early semi-convergence, i.e., SIRT can overfit to noise in the reconstruction [4].

In this set-up no paired noisy images are available for image-to-image denoising, and, as shown in section 5.5.1, self-supervised denoising in the reconstruction domain is challenging in the case of SIRT. Blind-spot denoising, on the other hand, is still able to take advantage of the large amount of imaged bubbles using the similarity within the data of an experiment. Moreover, for this data set, it is more computationally efficient than deep-learned denoising in reconstruction space, as the 3-tuple projections has a lower dimensionality than the volume (each timeframe has 3×1548×550 pixels and each volume has 1548×550×550 voxels, respectively).

In figure 5.7 we show a noisy and blind-spot denoised bubble of a single timeframe, after training on a subset of 1,450 timeframes of single-bubble-injection experimental data. We estimated the kernel on a static part of the experimental data, and used the same direct deconvolution and network architecture as in section 5.4.3, but preprocessed the 3×1450 projections via a tailored preprocessing procedure to remove the PMMA cylinder (figure 5.2, Chapter 2). Qualitatively, the blind-spot denoiser is able to recover both sharp and smooth features of the bubbles well. Figure 5.8 visualizes the subsequent SIRT reconstructions. The noisy bubble is visible as a denser attenuation in a highly noisy field, and we were only able to visualize it by clipping the attenuation values to a narrow range. On the other hand, the denoised bubbles provide a much better contrast in the reconstruction, have gradual interfaces, and we expect will be better suited for further statistical analysis of the fluidized beds.

5.6 Discussion

Direct deconvolution is an efficient and effective approach enabling self-supervised denoising of real-world X-ray radiographs by reverting scintillator-induced correlations. For the deconvolution kernel, an empirically-obtained estimate of the scintillator point-response function has an implicit regularizing effect, as it balances the introduction of new correlations as checkerboard-artifacts with the removal of existing correlations. In this article, we demonstrated that the denoising potential of Noise2Self is restored, and that no ground truths or paired noisy training examples are required. In the results, we highlight its importance for projection-domain denoising via an example of real-world dynamic sparse-view CT.

Our approach tested well on raw data from Caesium-Iodine detectors in the presence of high photon counts, high noise due to scatter, and low anisotropic additive noise. We expect this scenario to be representative for cone beam set-ups where the photon noise

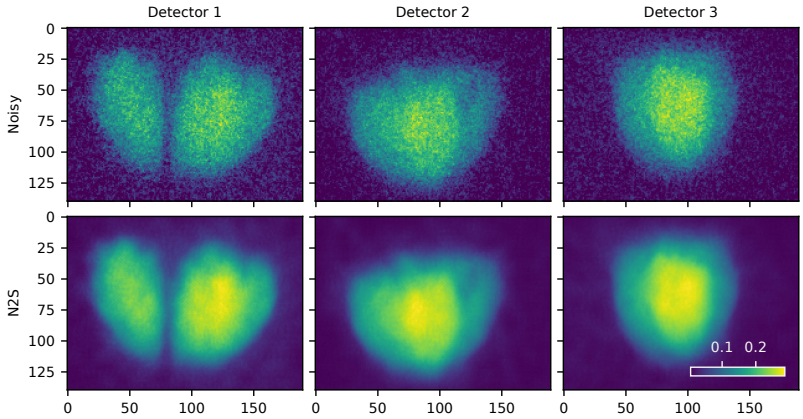


Figure 5.7: Top row: preprocessed projections (Chapter 2) of the single-bubble-injection experiment at timeframe 238/4500 (view is restricted to the bubble). Detector 1 shows that the bubble has split in two. **Bottom row**: Noise2Self network outputs.

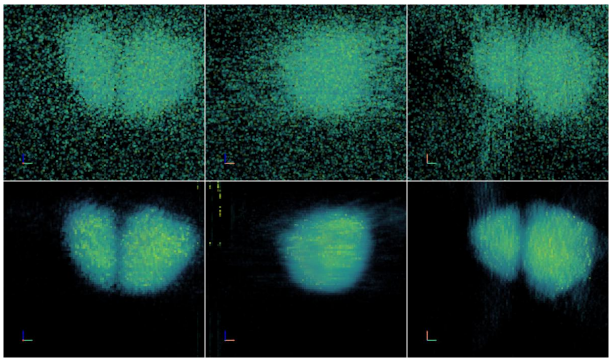


Figure 5.8: Ultra-sparse SIRT reconstructions of the noisy and denoised projections displayed in figure 5.7. **Left**: Side view along the projection axis of detector 1. **Middle**: Side view along the horizontal axis that is orthogonal to the projection axis. **Right**: Top view.

is dominant and the scintillator blur approximately uniform. It would be interesting to benchmark the method in the limits of different noise conditions [161, 170], such as the low-intensity radiation used in biomedical microscopy, monochromatic or filtered X-ray energy spectra, and different detectors, e.g., thick monolithic scintillators [142].

In algorithm 2, we sampled correlations from radiographs of a static object, and used these to find a suitable deconvolution kernel. While this approach is efficient, its estimation error may prove too large for certain situations, e.g., with limited quality sampling data, for detectors with a large blur radius, or for set-ups that utilize a varying tube current [138]. In this case, solving the ground truth λ and kernel $\tilde{h}$ simultaneously, e.g., with a blind deconvolution algorithm, could be investigated. At the same time, uniform deconvolution becomes less effective in the presence of anisotropic and nonlinear noise. In order to move to a more sophisticated model, such as non-uniform deconvolution, it is necessary to examine the correlation structures in real-world radiographs in more detail, and in particular the way that they depend on the signal.

An open question is if kernel estimation and deconvolution can be integrated together with self-supervised learning, for example as an additional loss penalty [45]. Doing so is not straightforward: When both the deconvolved and convolved data are present in the loss, e.g., as a "Deconvolved2Convolved" strategy, neural networks can collapse into a convolution, in which case the noise is still propagated. Further investigation is also needed to see if pre-removal of non-Poisson noise prior to deconvolution, or learned deconvolution [175], could prove advantageous.

Acknowledgements

We thank Evert Wagner, Sophia Podber, and Luis Portela of Delft University of Technology for the radiograph data of polystyrene phantoms and single-bubble-injected fluidized beds. This work was supported by the Dutch Research Council (NWO, project numbers 613.009.106, 613.009.116, and VI.Vidi.223.059).

5.7 Appendix: Source code availability

Software is available at `github.com/adriaangraas/scintillatordecorrelator`. The radiographs of experimental phantoms (section 5.4.2) have been published on Zenodo [176]. Data of fluidized beds is available upon reasonable request.

5.8 Appendix: Estimation of the deconvolution kernel

Statistical moments of convoluted radiographs We consider radiographs given as $I := H\hat{I} + \varepsilon$, with $\hat{I} \sim \text{Poisson}(\Lambda)$ and anisotropic $\varepsilon \sim \mathcal{N}(0, \Sigma)$ with $\Sigma := \text{diag}(\sigma_1^2, \ldots, \sigma_{N_u N_v}^2)$. Their mean estimates the clean convoluted image:

$$\mathbb{E}[I] = \mathbb{E}[h] \circledast \mathbb{E}[\hat{I}] + \mathbb{E}[\varepsilon] = h \circledast \lambda. \tag{5.16}$$

Linearity of the covariance with independence of $\hat{I}$ and ε gives

$$\operatorname{Cov}(H\hat{I}+\varepsilon) = H\operatorname{Cov}(\hat{I})H^{\mathsf{T}} + \operatorname{Cov}(\varepsilon) = H\Lambda H^{\mathsf{T}} + \Sigma, \tag{5.17}$$

and each (i,j)-th entry can be written as

$$\begin{aligned}\operatorname{Cov}(H\hat{I}+\varepsilon)_{ij} &= \textstyle\sum_k \sum_l H_{il}\Lambda_{lk}H^{\mathsf{T}}_{kj} + \Sigma_{ij} && (5.18)\\ &= \textstyle\sum_k h_{i-k}\lambda_k h_{j-k} + \Sigma_{ij} && (5.19)\\ &= \textstyle\sum_k h_k h_{j-i+k}\lambda_{i-k} + \Sigma_{ij}, && (5.20)\end{aligned}$$

where the second equality uses the definition of convolution, $H_{ij} = h_{i-j}$ and $H^{\mathsf{T}}_{ij} = h_{j-i}$, and the third equality shifts the indices.

5

Statistical moments of radiograph differences Let I and I' be independent observations. For the difference $I - I'$, the mean is

$$\mathbb{E}[I - I'] = \mathbb{E}[I] - \mathbb{E}[I'] = 0, \tag{5.21}$$

using equation (5.16), whereas the covariance is given by

$$\operatorname{Cov}(I - I') = \operatorname{Cov}(I) + \operatorname{Cov}(I') = 2\operatorname{Cov}(I). \tag{5.22}$$

Similarly, $\operatorname{Var}(I - I') = 2\operatorname{Var}(I)$.

Kernel estimation problem The goal is to find a deconvolution operator $\widetilde{H}^{+}$ to transform $H\hat{I} + \varepsilon$ into $\hat{I}$, i.e., an image with statistically independent noise. Equation equation (5.9) and equation (5.10) in section 5.3.2 explain that we would like $\widetilde{H}^{+}$ to perform a diagonalization of the covariance matrix:

$$H\Lambda H^{\mathsf{T}} + \Sigma = \widetilde{H}\Lambda\widetilde{H}^{\mathsf{T}}. \tag{5.23}$$

Rearranging the terms, and using equation (5.20), shows that the kernel $\tilde{h}$ of $\widetilde{H}$ must obtain

$$\sum_k (h_k h_{j-i+k} - \tilde{h}_k\tilde{h}_{j-i+k})\lambda_{i-k} + \Sigma_{ij} = 0 \tag{5.24}$$

for all elements (i,j). Achieving zero correlation, however, is not possible, as a solution $\tilde{h}$ cannot simultaneously cancel the signal (λ_{i-k}) and noise (Σ_{ij}) terms.

Correlation coefficients Algorithm 2 instead finds a minimizer of equation (5.24) via the use of correlation coefficients. Denote with

$$M := \frac{I - I'}{\sqrt{\mathrm{Var}(I - I')}}, \tag{5.25}$$

the variance-stabilized noise image (division is pixelwise). The covariance between M_i and M_j computes to

$$\begin{aligned}
\mathrm{Cov}(M_i, M_j) &= \frac{2\,\mathrm{Cov}(I_i, I_j)}{\sqrt{2\,\mathrm{Var}(I_i)}\sqrt{2\,\mathrm{Var}(I_j)}} && (5.26)\\
&\approx \frac{\lambda_i \sum_k h_k h_{j-i+k} + \Sigma_{ij}}{\lambda_i \sum_k h_k^2 + \sigma_i} && (5.27)\\
&= \frac{1}{\alpha_i} \sum_k h_k h_{j-i+k} + \Sigma_{ij}/\lambda_i && (5.28)\\
&=: \frac{1}{\alpha_i} \tilde{c}_j. && (5.29)
\end{aligned}$$

The first equality uses equation (5.22), the second equality applies equation (5.18) and uses $\lambda_i \approx \lambda_{i-k}$ to remove λ_{i-k} from the sum. The third and fourth equalities introduce the notation for the correlation coefficients and scaling parameters $\alpha_i = \sum_k h_k^2 + \sigma_i^2/\lambda_i$.

Under the smoothness assumption, $\lambda_i \approx \lambda_{i-k}$, it is straightforward to see that the coefficients $\tilde{c}_j$ provide a direct solution to equation equation (5.24). From here, finding a suitable deconvolution kernel entails finding a solution $\tilde{h}$ that best fits $\tilde{c}$. The accuracy of $\tilde{c}_j$ depends on the number of samples (cf. line 3 in algorithm 2), which in practice is improved by averaging $\tilde{c}_j$ over a set of pixels in the image.

Solution using the matrix square root Since the resulting kernel must be uniform, it suffices to formulate the optimization problem for a single pixel (i.e., one row of equation (5.23)). To do so, we form a small Toeplitz sample correlation matrix $\widetilde{C}$, using $\tilde{c}_j$ on the j-th diagonals. The matrix is then decomposed into convolution operators via the matrix square root, i.e., $\widetilde{H} \approx (\widetilde{H}\widetilde{H}^\intercal)^{1/2} = \widetilde{C}^{1/2}$ using a blocked Schur algorithm [164]. From here $\tilde{h}$ can be extracted as the first row or column.

Additional remarks

- *Smoothness assumption*: In equation equation (5.27) we have assumed $\lambda_i \approx \lambda_{i-k}$, which enables retrieving $\widetilde{H}$ via fast sampling (algorithm 2). The approach is justified for most radiographs, as their ground truth often consists of homogeneous or smooth regions, which bias the estimates $\tilde{c}_j$ towards a correct value. While we find our assumption to work well in practice (section 5.4.1), for situations where it would lead to an error outside of acceptable bounds, the sampling region can be restricted to a smaller or smoother region of the radiograph. We further reflect on this assumption in the discussion.

- *Scaling*: In X-ray imaging set-ups, radiograph intensities are calibrated to represent physical attenuation coefficients [3]. To ensure deconvolution does not interfere with

the calibration process, we normalize the found kernel $\tilde{h}$ such that its integral value equals one.

Conclusions and outlook

This thesis took a journey along different components of fast tomographic pipelines. It discussed a customized set-up for fast imaging, deep learning in real-time pipelines, software for fast algorithms, and a self-supervised method for the removal of noise. These components were often connected: For example, the fast set-up yielded sparse-view data (Chapter 2), and consequently only the slower SIRT algorithm was able to achieve sufficient quality. Similarly, in the real-time learning pipeline of Chapter 3, the underlying numerical implementation of the tomographic algorithm proved crucial for on-the-fly training of the neural network.

In this section, we will take an overarching view on the results, reflect on the contributions of the papers, and provide directions for future research.

X-ray CT set-ups In Chapter 2, the authors investigated an ultra-sparse set-up for imaging the phenomena of fluidized beds. Fluidized beds behave similar to fluids, e.g., voids in fluidized beds can disperse and merge, similar to bubbles in a liquid. The authors first addressed several difficulties with the construction of the set-up, such as its calibration procedure to retrieve the positions and orientations of three sources and detectors, and the temporal synchronization of the three detectors. For bubbling fluidized beds, the set-up also required a tailored referencing procedure in order to extract the bubbles from radiographs with fluctuating backgrounds of particles.

In this research, their main contribution was to enable *time-resolved and fully-3D* reconstruction of gas-solids beds. Despite the ultra-sparse geometry, the high-resolution flat-panel detectors provided sufficient information that allowed inferring bubble characteristics, such as their shape, location and density in the volume. For example, phantom experiments were conducted to demonstrate that the new method enables reliable estimation of bubble diameters. Compared to existing set-ups, which, e.g., only image in a small horizontal slice, or are not time-resolved, this set-up has the unique ability to look at large-scale dynamic evolution of the beds.

To achieve the sparse-angle reconstruction, the authors leveraged SIRT (cf. Chapter 1). This algebraic technique enables the integration of prior information, which to some extent absorbed the difficulties of the sparse-view geometry and fast acquisition. We found two

main factors to play a pivotal role in the reconstruction accuracy. The first was the usage of simple prior information, such as a masking region and box constraints, constraining the solution to the physically-attainable gas-solids ratios. The second was the amplitude of noise in the data. The most prominent source of noise was due to cross-scattering from the non-facing sources, and another was due to the granularity of particles in the data. Without mask or box constraints, or with high noise, SIRT cannot recover the smooth boundaries of bubbles. Instead, its solutions reflect the geometry of the set-up, and the bubbles become hexagonally-shaped in the horizontal plane. In real-world data, however, this effect was somewhat reduced, due to the smooth interfaces of bubbles: the SIRT solution only "hexagonalizes" uniform regions in the reconstruction.

One clear research direction is to study joined denoising-and-reconstruction technique in combination with the data retrieved from the set-up, as pursued in Chapter 5.2.2. Another straightforward way to further increase the accuracy of the reconstruction, is to extend the set-up with additional source-detector pairs. However, the existing set-up may also already be improved by changing its geometry, e.g., the sources and detectors could be aligned on a Cartesian coordinate system. A next direction for research would be to integrate (deep learned) priors about fluidized beds into the iterative method. Yet another would be to explore the temporal aspect of fluidized beds. Since bubbles are subject to morphological changes as they travel upwards, dynamic priors about the bubbles' positions or gas contents could be incorporated in the reconstruction process. However, bubbles in a bed borrow from, and release to, interstitial gas, and learning these complex dynamics may prove challenging, especially when the timesteps between reconstruction frames are large.

Real-time tomographic pipelines In Chapter 3, a neural network training strategy was introduced for real-time tomographic pipelines, such as are used within synchrotron beamlines. In these pipelines, off-the-shelf deep-learned algorithms (e.g. trained on pictures of photo cameras) do not generalize well due to the properties of experimental data. In this chapter, the authors pioneered a learning technique for on-the-fly reconstruction and training of neural networks, using a proof-of-concept implementation on intercepted radiographs of the RECAST3D pipeline software. They investigated different buffer sizes, fast and slow dynamics, and several variations of U-Net network architectures, on dissolving tablets from the FleX-ray laboratory with the Noise2Inverse denoising task.

In the research, it was found that pretrained CNNs (i.e., networks that are trained until convergence, on a hold-out part of the experimental data) cannot be expected to generalize well on out-of-distribution samples. While, for the FleX-ray experiments, such networks would generalize well for the next 12 seconds, the pretrained CNNs had diminishing performance under changing experimental conditions or diverging experimental dynamics. At the same time, networks that were trained for during the experiment could quickly anticipate changes in the data and deliver much better results on unseen data.

While the results of just-in-time learning are promising for synchrotron light sources, the idea is still in a conceptual stage. Even though customized software and patch-based

training made the idea feasible, the main challenge remains the efficiency of training CNNs on-line. The experiments in the article used a region-of-interest FDK, a relatively simple training task, relatively slow experimental dynamics, and small network architectures. These properties proved realistic for the Noise2Inverse task, but for extending them to high-resolution, fully-3D reconstructions, fast dynamics, or complex learning tasks, the approach would require further research and would likely need much more computational resources (e.g., parallelization over a cluster of GPUs).

At the same time, there exist many opportunities to further improve the concept. An interesting, more theoretical inclined direction, is that of on-line learning. In the proof-of-concept, a simple capacity queue was used to decide which selection of historic radiographs was used as training samples. The size of the queue was hand-picked. However, a more intelligent selection has large potential to improve generalization. For example: under rapidly changing dynamics, the training procedure should focus only on the most recently reconstructed samples. And, with slower dynamics, also older radiographs should be included to prevent catastrophic forgetting. A further research direction that could prove helpful for training tasks that are more complex than denoising, is to combine the just-in-time strategy with transfer learning. For example, one could train a neural network partially off-line, and then continuously fine-tune it during the experiment.

Software for tomographic projectors In Chapter 4, the authors introduced ASTRA KernelKit, a Python software package that enables easy customization of tomographic projectors. Within the software, "projectors" not only refer the computational operations of the X-ray transform, but also include, e.g., memory management on the CPU and GPU, as well as preparatory calculations for the projection and volume geometries. The software was initially developed for Chapter 3 to facilitate the generation of reconstruction patches while the neural network training is ongoing, and then was then continued in its own project.

To motivate the new software package, the paper authors had first investigated several existing packages for Computed Tomography. Many, if not all, of those packages assumed a "user philosophy", that is, they were built for users to launch efficient pre-implemented CT algorithms. The latent CUDA kernels of these packages were similar in terms of efficiency, since they all followed best-practices for single three-dimensional reconstruction volumes. Unfortunately, the philosophy is problematic for researchers that need to experiment with custom algorithms, for example for neural networks or high-dimensional problems. KernelKit therefore instead takes a "researcher philosophy": Both Python and CUDA operations are easily accessible, allowing researchers to extend or modify the core routines to fit their needs. In the results, the software was demonstrated for tomographic pipelines. The examples were: CUDA graphs for neural networks, slice-based reconstruction with RECAST3D (Chapter 3), and kernel tuning for the fluidized-bed set-up (Chapter 2).

There are two main research directions for this project. One is the project itself, and the other is the utilization of KernelKit in other research projects. Within the project itself, there are several possibilities for extensions. Currently, KernelKit only implements the

standard ASTRA kernels. However, it could benefit from recent developments for CT, such as the distance-driven projectors. Another technical direction is to support more devices, for example, AMD ROCm architectures, or NVIDIA Hopper or Grace architectures. The possible of run-time compilation also allow software features that accommodate to different reconstruction problems: For example, KernelKit could automatically compile a kernel that skips all computations inside a masked area of the volume.

The second research direction is in projects that have become possible or easier with KernelKit. One particular interesting direction is kernel tuning. Currently, KernelKit compiles a standard kernel when the user does not specify parameters, such as the number of X, Y, and Z CUDA threads that determine how the workload is divided. Oftentimes, tomographic algorithms can take several minutes or longer to compute, with the projectors taking the majority of the computation time. Probing the parameter space before of the start of the algorithm can take as little as a few seconds, and can increase the efficiency of the algorithm or lower its power consumption.

Self-supervised denoising of radiographs Lastly, in Chapter 5, the paper authors sought a solution for self-supervised blind-spot denoising of radiographs. As explained in Chapter 1 and 5, blind-spot denoisers work for noisy images without paired ground truths, but unlike zero-shot denoisers, e.g., the Deep Image Prior, can still take information from large data sets of single noisy images.

The motivation for this research originated from the set-up of Chapter 2, which provided a data set comprising many noisy radiographs of bubbles. At that time, no self-supervised learning techniques existed that could take advantage of this noisy data. They required, for example, paired noisy radiographs, radiographs from nearby angles, repeated patches within a radiograph (similar to BM3D), or downsampling, to make sure that multiple independent noisy samples could be extracted from the data. In the research, the authors investigated why the most promising class of self-supervised denoisers, the blind-spot denoisers, did not work for the radiographs from X-ray detectors, and found the cause to be due to correlated photon fluctuations inside scintillator detectors. Since correlations were isotropic for real-world Caesium-Iodine scintillator detectors, they could be described by a uniform convolution. From here, a simple sampling-decorrelation workflow was derived, and the method tested successfully for the bubble radiographs and the SIRT reconstruction outlined in Chapter 2.

In this research, there are again two promising continuations. The first is a further improvement of the denoising workflow, i.e., as outlined in the conclusions at the end of Chapter 5: It can be tested for different different set-ups and detectors (e.g. monolithic detectors, different scintillator crystals), and it may be possible to suppress the side-effects of additive Gaussian noise. A possible improvement could be to extend the model to nonuniform deconvolution as to further reduce model mismatch (e.g. for heterogeneous pixel technology, or Swank noise).

An open question is if and how the denoising framework can be further integrated with with self-supervised tomographic reconstruction. In our work, it was considered as a pre-

processing technique before sparse-view reconstruction. Preprocessing can be more memory-efficient compared to, e.g., iterative learned techniques, such as the learned primal-dual. On the other hand, for situations where the learned 3D reconstruction would be possible (small volumes, or 2D parallel beam), an opportunity could be to integrate blind-spot denoising in the reconstruction. This would have the advantage that neural network filters work on reconstructed data (where the image features are local), while the loss is taken on the radiographs (where the noise is local).

Bibliography

[1] Arati S. Panchbhai. "Wilhelm Conrad Röntgen and the discovery of X-rays: Revisited after centennial". In: *Journal of Indian Academy of Oral Medicine and Radiology* 27.1 (2015). ISSN: 0972-1363.

[2] Raymond A Schulz, Jay A Stein, and Norbert J Pelc. "How CT happened: the early development of medical computed tomography". en. In: *J Med Imaging (Bellingham)* 8.5 (2021), p. 052110.

[3] T.M. Buzug. *Computed Tomography: From Photon Statistics to Modern Cone-Beam CT*. Springer, 2008. ISBN: 9783540394075.

[4] Per Christian Hansen, Jakob Jørgensen, and William R. B. Lionheart. *Computed Tomography: Algorithms, Insight, and Just Enough Theory*. Ed. by Per Christian Hansen, Jakob Jørgensen, and William R. B. Lionheart. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2021.

[5] Avinash C. Kak and Malcolm Slaney. *Principles of computerized tomographic imaging*. Philadelphia: Society for Industrial and Applied Mathematics, 2001.

[6] Lee Feldkamp, L. C. Davis, and James Kress. "Practical Cone-Beam Algorithm". In: *Journal of the Optical Society of America* 1 (1984), pp. 612–619.

[7] Philippe Després and Xun Jia. "A review of GPU-based medical image reconstruction". In: *Physica Medica* 42 (2017), pp. 76–92. ISSN: 1120-1797.

[8] Jens Gregor and Thomas Benson. "Computational analysis and improvement of SIRT". In: *IEEE transactions on medical imaging* 27.7 (2008), pp. 918–924.

[9] Sophia Bethany Coban, Felix Lucka, Willem Jan Palenstijn, Denis Van Loo, and Kees Joost Batenburg. "Explorative Imaging and Its Implementation at the FleX-ray Laboratory". In: *Journal of Imaging* 6.4 (2020). ISSN: 2313-433X.

[10] Michal Vopalensky, Petr Koudelka, Jan Sleichrt, Ivana Kumpova, Matej Borovinsek, Matej Vesenjak, and Daniel Kytyr. "Fast 4D On-the-Fly Tomography for Observation of Advanced Pore Morphology (APM) Foam Elements Subjected to Compressive Loading". In: *Materials* 14.23 (2021). ISSN: 1996-1944.

[11] Rachael M. Wood, Dirk E. Schut, Anna K. Trull, Leo F.M. Marcelis, and Rob E. Schouten. "Detecting internal browning in apple tissue as determined by a single CT slice in intact fruit". In: *Postharvest Biology and Technology* 211 (2024), p. 112802. ISSN: 0925-5214.

[12] Jan-Willem Buurlage, Holger Kohr, Willem Jan Palenstijn, and Joost Batenburg. "Real-time quasi-3D tomographic reconstruction". In: *Measurement Science and Technology* 29.6 (2018).

[13] Martin Berger, Qiao Yang, and Andreas Maier. “X-ray Imaging”. en. In: *Medical Imaging Systems: An Introductory Guide*. Cham (CH): Springer, 2018, pp. 119–145.

[14] Saeed Izadi, Darren Sutton, and Ghassan Hamarneh. “Image denoising in the deep learning era”. In: *Artificial Intelligence Review* 56 (Nov. 2022), pp. 1–46.

[15] Samuel W. Hasinoff. “Photon, Poisson Noise”. In: *Computer Vision: A Reference Guide*. Springer US, 2014, pp. 608–610. ISBN: 978-0-387-31439-6.

[16] Buda Bajić, Johannes A. J. Huber, Benedikt Neyses, Linus Olofsson, and Ozan Öktem. *Reconstruction for Sparse View Tomography of Long Objects Applied to Imaging in the Wood Industry*. 2024.

[17] Viktor V. Nikitin, Marcus Carlsson, Fredrik Andersson, and Rajmund Mokso. “Four-Dimensional Tomographic Reconstruction by Time Domain Decomposition”. In: *IEEE Transactions on Computational Imaging* 5.3 (2019), pp. 409–419.

[18] Fang Xu and K. Mueller. “A comparative study of popular interpolation and integration methods for use in computed tomography”. In: *3rd IEEE International Symposium on Biomedical Imaging: Nano to Macro, 2006.* 2006, pp. 1252–1255.

[19] Peter M. Joseph. “An Improved Algorithm for Reprojecting Rays through Pixel Images”. In: *IEEE Transactions on Medical Imaging* 1.3 (1982), pp. 192–196.

[20] Eric Papenhausen, Ziyi Zheng, and Klaus Mueller. “GPU-Accelerated Back-Projection Revisited : Squeezing Performance by Careful Tuning”. In: 2011.

[21] Yiqiu Dong, Per Christian Hansen, Michiel E. Hochstenbach, and Nicolai André Brogaard Riis. “Fixing Nonconvergence of Algebraic Iterative Reconstruction with an Unmatched Backprojector”. In: *SIAM Journal on Scientific Computing* 41.3 (2019), A1822–A1839.

[22] G.L. Zeng and G.T. Gullberg. “Unmatched projector/backprojector pairs in an iterative reconstruction algorithm”. In: *IEEE Transactions on Medical Imaging* 19.5 (2000), pp. 548–555.

[23] Wim Van Aarle et al. “Fast and flexible X-ray tomography using the ASTRA toolbox”. In: *Optics express* 24.22 (2016), pp. 25129–25147.

[24] Ander Biguri, Manjit Dosanjh, Steven Hancock, and Manuchehr Soleimani. “TIGRE: a MATLAB-GPU toolbox for CBCT image reconstruction”. In: *Biomedical Physics & Engineering Express* 2.5 (2016), p. 055010.

[25] Daniël M. Pelt, Doğa Gürsoy, Willem Jan Palenstijn, Jan Sijbers, Francesco De Carlo, and Kees Joost Batenburg. “Integration of TomoPy and the ASTRA toolbox for advanced processing and reconstruction of tomographic synchrotron data”. In: *Journal of Synchrotron Radiation* 23.3 (2016), pp. 842–849.

[26] Allard Hendriksen, Dirk Schut, Willem Jan Palenstijn, Nicola Viganò, Jisoo Kim, Daniël Pelt, Tristan van Leeuwen, and K. Joost Batenburg. “Tomosipo: Fast, Flexible, and Convenient 3D Tomography for Complex Scanning Geometries in Python”. In: *Optics Express* (2021). ISSN: 1094-4087.

[27] Robert A Bridges, Neena Imam, and Tiffany M Mintz. “Understanding GPU power: A survey of profiling, modeling, and simulation methods”. In: *ACM Computing Surveys (CSUR)* 49.3 (2016), pp. 1–27.

[28] Richard Schoonhoven, Bram Veenboer, Ben Van Werkhoven, and K Joost Batenburg. "Going green: optimizing GPUs for energy efficiency through model-steered auto-tuning". In: *2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE. 2022, pp. 48–59.

[29] Simon Arridge, Peter Maass, Ozan Öktem, and Carola-Bibiane Schönlieb. "Solving inverse problems using data-driven models". In: *Acta Numerica* 28 (2019), pp. 1–174.

[30] Saiprasad Ravishankar, Jong Chul Ye, and Jeffrey A. Fessler. "Image Reconstruction: From Sparsity to Data-Adaptive Methods and Machine Learning". In: *Proceedings of the IEEE* 108.1 (2020), pp. 86–109.

[31] Gregory Ongie, Ajil Jalal, Christopher A. Metzler, Richard G. Baraniuk, Alexandros G. Dimakis, and Rebecca Willett. "Deep Learning Techniques for Inverse Problems in Imaging". In: *IEEE Journal on Selected Areas in Information Theory* 1.1 (2020), pp. 39–56.

[32] Jong Chul Ye, Yonina C. Eldar, and Michael Unser. *Deep Learning for Biomedical Image Reconstruction*. Cambridge University Press, 2023.

[33] Yong Huang, Nan Zhang, and Qun Hao. "Real-time noise reduction based on ground truth free deep learning for optical coherence tomography". In: *Biomed. Opt. Express* 12.4 (2021), pp. 2027–2040.

[34] Aniket Tekawade et al. "Real-time porosity mapping and visualization for synchrotron tomography". In: *TechRxiv* (2022).

[35] Richard Schoonhoven, Jan-Willem Buurlage, Daniël Pelt, and Joost Batenburg. "Real-time segmentation for tomographic imaging". In: *IEEE 30th International Workshop on Machine Learning for Signal Processing*. 2020, pp. 1–6.

[36] Julius Berner, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. "The Modern Mathematics of Deep Learning". In: *CoRR* abs/2105.04026 (2021).

[37] René Vidal, Joan Bruna, Raja Giryes, and Stefano Soatto. "Mathematics of Deep Learning". In: *CoRR* abs/1712.04741 (2017).

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015.

[39] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. "Noise2Noise: Learning Image Restoration without Clean Data". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 2965–2974.

[40] Allard A. Hendriksen, Daniel M. Pelt, and K. Joost Batenburg. "Noise2Inverse: Self-Supervised Deep Convolutional Denoising for Tomography". In: *IEEE Transactions on Computational Imaging* 6 (2020), pp. 1320–1335.

[41] Joshua Batson and Loic Royer. *Noise2Self: Blind Denoising by Self-Supervision*. 2019.

[42] Alexander Krull, Tim-Oliver Buchholz, and Florian Jug. "Noise2Void - Learning Denoising From Single Noisy Images". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019, pp. 2124–2132.

[43] Samuli Laine, Tero Karras, Jaakko Lehtinen, and Timo Aila. *High-Quality Self-Supervised Deep Image Denoising*. 2019.

[44] Alexander Krull, Tomáš Vičar, Mangal Prakash, Manan Lalit, and Florian Jug. "Probabilistic Noise2Void: Unsupervised Content-Aware Denoising". In: *Frontiers in Computer Science* 2 (Feb. 2020). ISSN: 2624-9898.

[45] Yaochen Xie, Zhengyang Wang, and Shuiwang Ji. *Noise2Same: Optimizing A Self-Supervised Bound for Image Denoising*. 2020.

[46] Adriaan B.M. Graas, Evert C. Wagner, Tristan van Leeuwen, J. Ruud van Ommen, K. Joost Batenburg, Felix Lucka, and Luis M. Portela. "X-ray tomography for fully-3D time-resolved reconstruction of bubbling fluidized beds". In: *Powder Technology* 434 (2024), p. 119269. ISSN: 0032-5910.

[47] D. Geldart. "Types of gas fluidization". In: *Powder Technology* 7.5 (1973), pp. 285–292. ISSN: 0032-5910.

[48] J.R. Van Ommen and R.F. Mudde. "Measuring the Gas-Solids Distribution in Fluidized Beds – A Review". In: *International Journal of Chemical Reactor Engineering* 6 (2008), pp. 1–29.

[49] Junli Xue, Muthanna Al-Dahhan, and M.P. Dudukovic. "Bubble Velocity, Size, and Interfacial Area Measurements in a Bubble Column by Four-Point Optical Probe". In: *Fluid Mechanics and Transport Phenomena* 54 No.2 (2007), pp. 350–363.

[50] F. Schillinger, T. J. Schildhauer, S. Maurer, E. Wagner, R. F. Mudde, and J. R. van Ommen. "Generation and evaluation of an artificial optical signal based on X-ray measurements for bubble characterization in fluidized beds with vertical internals". In: *International Journal of Multiphase Flow* 107 (2018), pp. 16–32.

[51] Haigang Wang and Wuqiang Yang. "Application of electrical capacitance tomography in circulating fluidised beds – A review". In: *Applied Thermal Engineering* 176 (2020), p. 115311. ISSN: 1359-4311.

[52] D. J. Parker, C. J. Broadbent, P. Fowles, M. R. Hawkesworth, and P. McNeil. "Positron emission particle tracking - a technique for studying flow within engineering equipment". In: *Nuclear Inst.and Methods in Physics Research, A* 326.3 (1993), pp. 592–607.

[53] R. F. Mudde. "Time-resolved X-ray tomography of a fluidized bed". In: *Powder Technology* 199.1 (2010), pp. 55–59.

[54] G.C. Brouwer, E.C. Wagner, J.R. Van Ommen, and R.F. Mudde. "Effects of pressure and fines content on bubble diameter in a fluidized bed studied using fast X-ray tomography". In: *Chemical Engineering Journal* 207-208 (2012), pp. 711–717.

[55] A. Helmi, E. C. Wagner, F. Gallucci, M. van Sint Annaland, J. R. van Ommen, and R. F. Mudde. "On the hydrodynamics of membrane assisted fluidized bed reactors using X-ray analysis". In: *Chemical Engineering and Processing: Process Intensification* 122 (2017), pp. 508–522.

[56] Saad Jahangir, Evert C. Wagner, Robert F. Mudde, and Christian Poelma. "Void fraction measurements in partial cavitation regimes by X-ray computed tomography". In: *International Journal of Multiphase Flow* 120.103085 (2019).

[57] M. M. Mandalahalli, E. C. Wagner, L. M. Portela, and R. F. Mudde. "Electrolyte effects on recirculating dense bubbly flow: An experimental study using X-ray imaging". In: *AIChE Journal* 66.1 (2020).

[58] M. Wang. *Industrial Tomography: Systems and Applications*. Woodhead Publishing Series in Electronic and Optical Materials. Elsevier Science, 2022. ISBN: 9780128233078.

[59] Geir Anton Johansen, Uwe Hampel, and Bjørn Tore Hjertaker. "Flow imaging by high speed transmission tomography". In: *Applied Radiation and Isotopes* 68.4 (2010). The 7th International Topical Meeting on Industrial Radiation and Radio isotope Measurement Application (IRRMA-7), pp. 518–524. ISSN: 0969-8043.

[60] Vasile Neculaes, Peter Edic, Mark Frontera, Antonio Caiafa, Ge Wang, and Bruno De Man. "Multisource X-ray and CT: Lessons Learned and Future Outlook". In: *IEEE Access* 2 (2015), pp. 1–1.

[61] Thomas G. Flohr et al. "First performance evaluation of a dual-source CT (DSCT) system". In: *European Radiology* 16.2 (2006), pp. 256–268. ISSN: 1432-1084.

[62] Weiwen Wu et al. *Stationary Multi-source AI-powered Real-time Tomography (SMART)*. 2022.

[63] S. B. Kumar, D. Moslemian, and M.P. Dudukovic. "A γ-ray tomographic scanner for imaging voidage distribution in two-phase flow systems". In: *Flow Measurement and Instrumentation* 6.1 (1995), pp. 61–73.

[64] J. Chen, P. Gupta, S. Degaleesan, M. H. Al-Dahhan, M. P. Duduković, and B. A. Toseland. "Gas holdup distributions in large-diameter bubble columns measured by computed tomography". In: *Flow Measurement and Instrumentation* 9.2 (1998), pp. 91–101.

[65] Uwe Hampel et al. "A Review on Fast Tomographic Imaging Techniques and Their Potential Application in Industrial Process Control". In: *Sensors* 22.6 (2022). ISSN: 1424-8220.

[66] F. Guillard, B. Marks, and I. Einav. "Dynamic X-ray radiography reveals particle size and shape orientation fields during granular flow". In: *Scientific Reports* 7.1 (2017).

[67] M. Bieberle, F. Barthel, H. -. Menz, H. -. Mayer, and U. Hampel. "Ultrafast three-dimensional X-ray computed tomography". In: *Applied Physics Letters* 98.3 (2011).

[68] C. M. Boyce, A. Penn, M. Lehnert, K. P. Pruessmann, and C. R. Müller. "Magnetic resonance imaging of interaction and coalescence of two bubbles injected consecutively into an incipiently fluidized bed". In: *Chemical Engineering Science* 208 (2019).

[69] Fei Wang, Qussai Marashdeh, Aining Wang, and Liang-Shih Fan. "Electrical Capacitance Volume Tomography Imaging of Three-Dimensional Flow Structures and Solids Concentration Distributions in a Riser and a Bend of a Gas–Solid Circulating Fluidized Bed". In: *Industrial & Engineering Chemistry Research* 51.33 (2012), pp. 10968–10976.

[70] T.C. Chandrasekera et al. "A comparison of magnetic resonance imaging and electrical capacitance tomography: An air jet through a bed of particles". In: *Powder Technology* 227 (2012). Emerging Particle Technology, pp. 86–95. ISSN: 0032-5910.

[71] Alexander Penn, Takuya Tsuji, David O. Brunner, Christopher M. Boyce, Klaas P. Pruessmann, and Christoph R. Müller. "Real-time probing of granular dynamics with magnetic resonance". In: *Science Advances* 3.9 (2017), e1701879.

[72] A.J. Wilkinson, E.W. Randall, J.J. Cilliers, D.R. Durrett, T. Naidoo, and T. Long. "A 1000-measurement frames/second ERT data capture system with real-time visualization". In: *IEEE Sensors Journal* 5.2 (2005), pp. 300–307.

[73] Jennifer L. Mueller and Samuli Siltanen. *Linear and Nonlinear Inverse Problems with Practical Applications*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2012.

[74] P. Russo. *Handbook of X-ray Imaging: Physics and Technology*. Series in Medical Physics and Biomedical Engineering. CRC Press, 2017. ISBN: 9781498741545.

[75] Jorge J. Moré. “The Levenberg-Marquardt algorithm: Implementation and theory”. In: *Numerical Analysis*. Ed. by G. A. Watson. Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 105–116. ISBN: 978-3-540-35972-2.

[76] Mamoru Ishii and Takashi Hibiki. *Thermo-fluid dynamics of two-phase flow*. Springer Science & Business Media, 2011.

[77] H Enwald, E Peirano, and A.-E Almstedt. “Eulerian two-phase flow theory applied to fluidization”. In: *International Journal of Multiphase Flow* 22 (1996), pp. 21–66. ISSN: 0301-9322.

[78] Andreas Hauptmann, Ozan Öktem, and Carola Schönlieb. “Image reconstruction in dynamic inverse problems with temporal models”. In: *Handbook of Mathematical Models and Algorithms in Computer Vision and Imaging: Mathematical Imaging and Vision* (2021), pp. 1–31.

[79] RF Mudde, PRP Bruneau, and THJJ Van der Hagen. “Time-resolved γ-densitometry imaging within fluidized beds”. In: *Industrial & Engineering Chemistry Research* 44.16 (2005), pp. 6181–6187.

[80] Wim van Aarle et al. “Fast and flexible X-ray tomography using the ASTRA toolbox”. In: *Opt. Express* (Oct. 2016).

[81] John Grace, Xiaotao Bi, and Naoko Ellis. “Essentials of Fluidization Technology”. In: John Wiley & Sons, Ltd, 2020. ISBN: 9783527699483.

[82] V. Verma, J. T. Padding, N. G. Deen, J. A. M. Hans Kuipers, F. Barthel, M. Bieberle, M. Wagner, and U. Hampel. “Bubble dynamics in a 3-D gas-solid fluidized bed using ultrafast electron beam X-ray tomography and two-fluid model”. In: *AIChE Journal* 60.5 (2014), pp. 1632–1644.

[83] J. Saayman, W. Nicol, J. R. Van Ommen, and R. F. Mudde. “Fast X-ray tomography for the quantification of the bubbling-, turbulent- and fast fluidization-flow regimes and void structures”. In: *Chemical Engineering Journal* 234 (2013), pp. 437–447.

[84] R. F. Mudde. “Bubbles in a fluidized bed: A fast X-ray scanner”. In: *AIChE Journal* 57.10 (2011), pp. 2684–2690.

[85] R F Mudde, J Alles, and T H J J van der Hagen. “Feasibility study of a time-resolving x-ray tomographic system”. In: *Measurement Science and Technology* 19.8 (2008), p. 085501.

[86] Adriaan Graas, Sophia Bethany Coban, K. Joost Batenburg, and Felix Lucka. “Just-in-time deep learning for real-time X-ray Computed Tomography”. In: *Scientific Reports* 13.1 (2023), p. 20070. ISSN: 2045-2322.

[87] Hans Vanrompay, Jan-Willem Buurlage, Daniël M. Pelt, Vished Kumar, Xiaolu Zhuo, Luis M. Liz-Marzán, Sara Bals, and K. Joost Batenburg. “Real-Time Reconstruction of Arbitrary Slices for Quantitative and In Situ 3D Characterization of Nanoparticles”. In: *Particle & Particle Systems Characterization* 37.7 (2020), p. 2000073.

[88] Federica Marone et al. “Time Resolved in situ X-Ray Tomographic Microscopy Unraveling Dynamic Processes in Geologic Systems”. In: *Frontiers in Earth Science* 7 (2020), p. 346. ISSN: 2296-6463.

[89] Philip J. Withers et al. “X-ray computed tomography”. In: *Nature Reviews Methods Primers* 1.1 (2021), p. 18. ISSN: 2662-8449.

[90] Viktor Nikitin, Aniket Tekawade, Anton Duchkov, Pavel Shevchenko, and Francesco De Carlo. "Real-time streaming tomographic reconstruction with on-demand data capturing and 3D zooming to regions of interest". In: *Journal of Synchrotron Radiation* 29.3 (2022), pp. 816–828.

[91] Jonathan Schwartz et al. "Real-Time 3D Analysis During Tomographic Experiments on tomviz". In: *Microscopy and Microanalysis* 27.S1 (2021), pp. 2860–2862.

[92] Jan-Willem Buurlage, Federica Marone, Daniël M. Pelt, Willem Jan Palenstijn, Marco Stampanoni, K. Joost Batenburg, and Christian M. Schlepütz. "Real-time reconstruction and visualisation towards dynamic feedback control during time-resolved tomography experiments at TOMCAT". In: *Scientific Reports* 9.1 (2019), p. 18379. ISSN: 2045-2322.

[93] Federica Marone, Alain Studer, Heiner Billich, Leonardo Sala, and Marco Stampanoni. "Towards on-the-fly data post-processing for real-time tomographic imaging at TOMCAT". In: *Advanced Structural and Chemical Imaging* 3.1 (2017), p. 1. ISSN: 2198-0926.

[94] Jonathan Schwartz, Huihuo Zheng, Marcus Hanwell, Yi Jiang, and Robert Hovden. "Dynamic Compressed Sensing for Real-time Tomographic Reconstruction". In: *Microscopy and Microanalysis* 26.S2 (2020), pp. 2462–2465. ISSN: 1435-8115.

[95] Bernhard Plank, R. Helmus, Maximilian Gschwandtner, Roland Hinterhölzl, and Johann Kastner. "In-Situ observation of bubble formation in neat resin during the curing process by means of X-ray computed tomography". In: 2016.

[96] Jaianth Vijayakumar, Niloofar Moazami Goudarzi, Guy Eeckhaut, Koen Schrijnemakers, Veerle Cnudde, and Matthieu N. Boone. "Characterization of Pharmaceutical Tablets by X-ray Tomography". In: *Pharmaceuticals* 16.5 (2023). ISSN: 1424-8247.

[97] Pieter Hintjens. *ZeroMQ: messaging for many applications.* " O'Reilly Media, Inc.", 2013.

[98] Bernt Øksendal. *Stochastic Differential Equations: An Introduction with Applications (Universitext).* 6th. Springer, 2014. ISBN: 3540047581.

[99] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. *Online Deep Learning: Learning Deep Neural Networks on the Fly.* 2017.

[100] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. *Gradient based sample selection for online continual learning.* 2019.

[101] Zahid Ali Siddiqui and Unsang Park. "Progressive Convolutional Neural Network for Incremental Learning". In: *Electronics* 10.16 (2021), p. 1879.

[102] Guanyu Zhou, Kihyuk Sohn, and Honglak Lee. "Online incremental feature learning with denoising autoencoders". In: *Artificial intelligence and statistics.* PMLR. 2012, pp. 1453–1461.

[103] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. *What is being transferred in transfer learning?* 2021.

[104] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. *A Comprehensive Survey of Continual Learning: Theory, Method and Application.* 2023.

[105] Antonin Chambolle. "An Algorithm for Total Variation Minimization and Applications". In: *Journal of Mathematical Imaging and Vision* 20.1 (2004), pp. 89–97. ISSN: 1573-7683.

[106] Richard Schoonhoven, Alexander Skorikov, Willem Jan Palenstijn, Daniël M. Pelt, Allard A. Hendriksen, and K. Joost Batenburg. "How auto-differentiation can improve CT workflows: classical algorithms in a modern framework". In: *Opt. Express* 32.6 (2024), pp. 9019–9041.

[107] Le Hou, Niki J. Parmar, Noam Shazeer, Xiaodan Song, Yeqing Li, and Youlong Cheng. "High Resolution Medical Image Analysis with Spatial Partitioning". In: *High Resolution Medical Image Analysis with Spatial Partitioning*. 2019.

[108] Arian Bakhtiarnia, Qi Zhang, and Alexandros Iosifidis. *Efficient High-Resolution Deep Learning: A Survey*. 2022.

[109] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. *What is the State of Neural Network Pruning?* 2020.

[110] Richard Schoonhoven, Allard A. Hendriksen, Daniël M. Pelt, and K. Joost Batenburg. *LEAN: graph-based pruning for convolutional neural networks by extracting longest chains*. 2022.

[111] Fabian Isensee, Philipp Kickingereder, Wolfgang Wick, Martin Bendszus, and Klaus H. Maier-Hein. *No New-Net*. 2019.

[112] Tianyuan Wang, Felix Lucka, and Tristan van Leeuwen. *Sequential Experimental Design for X-Ray CT Using Deep Reinforcement Learning*. 2023.

[113] Adriaan Graas, Willem Jan Palenstijn, Ben van Werkhoven, and Felix Lucka. "ASTRA KernelKit: GPU-accelerated projectors for Computed Tomography using CuPy". In: *Applied Mathematics for Modern Challenges* 2.1 (2024), pp. 70–92.

[114] Silvio Achilles et al. "GPU-accelerated coupled ptychographic tomography". In: *Developments in X-Ray Tomography XIV*. Ed. by Bert Müller and Ge Wang. Vol. 12242. International Society for Optics and Photonics. SPIE, 2022, 122420N.

[115] Nina Höflich and Oliver Pooth. "Studies on fast neutron imaging with a pixelated stilbene scintillator detector". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1040 (2022), p. 167211. ISSN: 0168-9002.

[116] J. S. Jørgensen et al. "Core Imaging Library - Part I: a versatile Python framework for tomographic imaging". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 379.2204 (2021), p. 20200192.

[117] Jonas Adler, Holger Kohr, and Ozan Öktem. *Operator Discretization Library*.

[118] Per Christian Hansen, Ken Hayami, and Keiichi Morikuni. "GMRES methods for tomographic reconstruction with an unmatched back projector". In: *Journal of Computational and Applied Mathematics* 413 (2022), p. 114352. ISSN: 0377-0427.

[119] Willem Jan Palenstijn, Joost Batenburg, and Jan Sijbers. "Performance improvements for iterative electron tomography reconstruction using graphics processing units (GPUs)". In: *Journal of Structural Biology* 176.2 (2011), pp. 250–253.

[120] Suren Chilingaryan, Evelina Ametova, Anreas Kopmann, and Alessandro Mirone. "Reviewing GPU architectures to build efficient back projection for parallel geometries". In: *Journal of Real-Time Image Processing* 17.5 (2020), pp. 1331–1373. ISSN: 1861-8219.

[121] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Second. Other Titles in Applied Mathematics. SIAM, 2003. ISBN: 978-0-89871-534-7.

[122] Rajmund Mokso et al. "GigaFRoST: the gigabit fast readout system for tomography". In: *Journal of Synchrotron Radiation* 24.6 (2017), pp. 1250–1259.

[123] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. *CUDA, release: 12.1*. 2023.

[124] Pieter Hijma, Stijn Heldens, Alessio Sclocco, Ben van Werkhoven, and Henri E. Bal. "Optimization Techniques for GPU Programming". In: *ACM Comput. Surv.* 55.11 (2023). ISSN: 0360-0300.

[125] Royud Nishino and Shohei Hido Crissman Loomis. "CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations". In: *31st confernce on neural information processing systems* 151.7 (2017).

[126] Pallets. *Jinja*. Version 3.1.2. Apr. 28, 2022.

[127] Folkert Bleichrodt, Tristan van Leeuwen, Willem Jan Palenstijn, Wim van Aarle, Jan Sijbers, and K. Joost Batenburg. "Easy implementation of advanced tomography algorithms using the ASTRA toolbox with Spot operators". In: *Numerical Algorithms* 71.3 (2016), pp. 673–697. ISSN: 1572-9265.

[128] Christopher Syben, Markus Michen, Bernhard Stimpel, Stephan Seitz, Stefan Ploner, and Andreas K. Maier. "Technical Note: PYRO-NN: Python reconstruction operators in neural networks". In: *Medical Physics* 46.11 (2019), pp. 5110–5115.

[129] Matteo Ronchetti. "TorchRadon: Fast Differentiable Routines for Computed Tomography". Preprint, 2020, arXiv2009.14788.

[130] Viktor Nikitin. "*TomocuPy* – efficient GPU-based tomographic reconstruction with asynchronous data processing". In: *Journal of Synchrotron Radiation* 30.1 (2023), pp. 179–191.

[131] Betsy A. Dowd, Graham H. Campbell, Robert B. Marr, Vivek V. Nagarkar, Sameer V. Tipnis, Lisa Axe, and D. Peter Siddons. "Developments in synchrotron X-ray computed microtomography at the National Synchrotron Light Source". In: *Developments in X-Ray Tomography II*. Ed. by Ulrich Bonse. Vol. 3772. International Society for Optics and Photonics. SPIE, 1999, pp. 224–236.

[132] David M. Beazley. "SWIG : An Easy to Use Tool for Integrating Scripting Languages with C and C++". In: *Fourth Annual USENIX Tcl/Tk Workshop (Fourth Annual USENIX Tcl/Tk Workshop)*. Monterey, CA: USENIX Association, 1996.

[133] Hui Shi, Yann Traonmilin, and Jean-François Aujol. "Compressive learning for patch-based image denoising". In: *SIAM Journal on Imaging Sciences* (2022).

[134] Fabian Altekrüger and Johannes Hertrich. "WPPNets and WPPFlows: The Power of Wasserstein Patch Priors for Superresolution". In: *SIAM Journal on Imaging Sciences* 16.3 (2023), pp. 1033–1067.

[135] Ryo Mashita, Wataru Yashiro, Daisuke Kaneko, Yasumasa Bito, and Hiroyuki Kishimoto. "High-speed rotating device for X-ray tomography with 10 ms temporal resolution". In: *Journal of Synchrotron Radiation* 28.1 (2021), pp. 322–326.

[136] Ben van Werkhoven. "Kernel Tuner: A search-optimizing GPU code auto-tuner". In: *Future Generation Computer Systems* 90 (2019), pp. 347–358.

[137] Adriaan Graas and Felix Lucka. "Scintillator decorrelation for self-supervised X-ray radiograph denoising". In: *Measurement Science and Technology* 36.6 (2025), p. 065415.

[138] Emilio Andreozzi, Antonio Fratini, Daniele Esposito, Mario Cesarelli, and Paolo Bifulco. "Toward a priori noise characterization for real-time edge-aware denoising in fluoroscopic devices". en. In: *Biomed Eng Online* (2021).

[139] Michael Elad, Bahjat Kawar, and Gregory Vaksman. *Image Denoising: The Deep Learning Revolution and Beyond – A Survey Paper –*. 2023.

[140] Wangmeng Zuo, Kai Zhang, and Lei Zhang. "Convolutional Neural Networks for Image Denoising and Restoration". In: *Denoising of Photographic Images and Video: Fundamentals, Open Challenges and New Trends*. Cham: Springer International Publishing, 2018, pp. 93–123. ISBN: 978-3-319-96029-6.

[141] Kyong Hwan Jin, Michael T. McCann, Emmanuel Froustey, and Michael Unser. "Deep Convolutional Neural Network for Inverse Problems in Imaging". In: *IEEE Transactions on Image Processing* 26.9 (2017), pp. 4509–4522.

[142] Zhehui Wang et al. "Needs, Trends, and Advances in Scintillators for Radiographic Imaging and Tomography". In: *IEEE Transactions on Nuclear Science* 70.7 (July 2023), pp. 1244–1280. ISSN: 1558-1578.

[143] A Bub, S Gondrom, M Maisl, N Uhlmann, and W Arnold. "Image blur in a flat-panel detector due to Compton scattering at its internal mountings". In: *Measurement Science and Technology* (2007).

[144] Marco Endrizzi, Piernicola Oliva, Bruno Golosio, and Pasquale Delogu. "CMOS APS detector characterization for quantitative X-ray imaging". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* (2013). ISSN: 0168-9002.

[145] Guillermo Avendaño Cervantes. *Technical Fundamentals of Radiology and CT*. 2053-2563. IOP Publishing, 2016. ISBN: 978-0-7503-1212-7.

[146] Jose Rocha and Senentxu Lanceros-Méndez. "Review on X-ray Detectors Based on Scintillators and CMOS Technology". In: *Recent Patents on Electrical Engineeringe* 4 (Jan. 2011), pp. 16–41.

[147] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Deep Image Prior". In: *International Journal of Computer Vision* 128.7 (Mar. 2020), pp. 1867–1888. ISSN: 1573-1405.

[148] Dongyu Yang et al. "Low-dose imaging denoising with one pair of noisy images". In: *Opt. Express* 31.9 (Apr. 2023), pp. 14159–14173.

[149] Xiaoya Chong, Min Cheng, Wenqi Fan, Qing Li, and Howard Leung. "M-Denoiser: Unsupervised image denoising for real-world optical and electron microscopy data". In: *Computers in Biology and Medicine* 164 (2023), p. 107308. ISSN: 0010-4825.

[150] Tao Huang, Songjiang Li, Xu Jia, Huchuan Lu, and Jianzhuang Liu. *Neighbor2Neighbor: Self-Supervised Denoising from Single Noisy Images*. 2021.

[151] Youssef Mansour and Reinhard Heckel. "Zero-Shot Noise2Noise: Efficient Image Denoising without any Data". In: *CoRR* abs/2303.11253 (2023).

[152] Dan Zhang, Fangfang Zhou, Yuwen Jiang, and Zhengming Fu. *MM-BSN: Self-Supervised Image Denoising for Real-World with Multi-Mask based on Blind-Spot Network*. 2023.

[153] Wooseok Lee, Sanghyun Son, and Kyoung Mu Lee. *AP-BSN: Self-Supervised Denoising for Real-World Images via Asymmetric PD and Blind-Spot Network*. 2022.

[154] Young-Joo Han and Ha-Jin Yu. *SS-BSN: Attentive Blind-Spot Network for Self-Supervised Denoising with Nonlocal Self-Similarity*. 2023.

[155] Xuanyu Tian, Zhuoya Dong, Xiyue Lin, Yue Gao, Hongjiang Wei, Yanhang Ma, Jingyi Yu, and Yuyao Zhang. *Zero-Shot Image Denoising for High-Resolution Electron Microscopy*. 2024.

[156] Ervin Dubaric, C. Fröjdh, Hans-Erik Nilsson, and Sture Petersson. "Resolution and noise properties of scintillator coated X-ray detectors". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 466 (June 2001), pp. 178–182.

[157] Melanie Freed, Subok Park, and Aldo Badano. “A fast, angle-dependent, analytical model of CsI detector response for optimization of 3D X-ray breast imaging systems”. en. In: *Med Phys* 37.6 (June 2010), pp. 2593–2605.

[158] Bo Kyung Cha, Youngjin Lee, and Kyuseok Kim. “Development of Adaptive Point-Spread Function Estimation Method in Various Scintillation Detector Thickness for X-ray Imaging”. In: *Sensors* 23.19 (2023). ISSN: 1424-8220.

[159] Robert K. Swank. “Absorption and noise in x-ray phosphors”. In: *Journal of Applied Physics* 44.9 (Sept. 1973), pp. 4199–4203. ISSN: 0021-8979.

[160] Yi Wang, Larry E Antonuk, Youcef El-Mohri, and Qihua Zhao. “A Monte Carlo investigation of Swank noise for thick, segmented, crystalline scintillators for radiotherapy imaging”. en. In: *Med Phys* 36.7 (July 2009), pp. 3227–3238.

[161] E N Gimenez et al. “Study of charge-sharing in MEDIPIX3 using a micro-focused synchrotron beam”. In: *Journal of Instrumentation* 6.01 (Jan. 2011), p. C01031.

[162] G. Dougherty and Z. Kawaf. “The point spread function revisited: image restoration using 2-D deconvolution”. In: *Radiography* 7.4 (Nov. 2001), pp. 255–262. ISSN: 1078-8174.

[163] K. Aditya Mohan, Robert M. Panas, and Jefferson A. Cuadra. “SABER: A Systems Approach to Blur Estimation and Reduction in X-Ray Imaging”. In: *IEEE Transactions on Image Processing* 29 (2020), pp. 7751–7764. ISSN: 1941-0042.

[164] Edvin Deadman, Nicholas J. Higham, and Rui Ralha. “Blocked Schur Algorithms for Computing the Matrix Square Root”. In: *Applied Parallel and Scientific Computing*. Ed. by Pekka Manninen and Per Öster. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 171–182. ISBN: 978-3-642-36803-5.

[165] Markku Makitalo and Alessandro Foi. “Optimal Inversion of the Anscombe Transformation in Low-Count Poisson Image Denoising”. In: *IEEE Transactions on Image Processing* 20.1 (2011), pp. 99–109.

[166] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. “Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising”. In: *IEEE Transactions on Image Processing* (2017). ISSN: 1941-0042.

[167] Daniël M. Pelt, Allard A. Hendriksen, and K. Joost Batenburg. “Foam-like phantoms for comparing tomography algorithms”. In: *Journal of Synchrotron Radiation* 29.1 (Jan. 2022), pp. 254–265.

[168] Adriaan B.M. Graas, Evert C. Wagner, Tristan van Leeuwen, J. Ruud van Ommen, K. Joost Batenburg, Felix Lucka, and Luis M. Portela. “X-ray tomography for fully-3D time-resolved reconstruction of bubbling fluidized beds”. In: *Powder Technology* 434 (2024), p. 119269. ISSN: 0032-5910.

[169] Nimu Yuan, Jian Zhou, and Jinyi Qi. “Half2Half: deep neural network based CT image denoising without independent reference data”. In: *Physics in Medicine & Biology* 65.21 (Nov. 2020), p. 215020.

[170] Kihwan Choi, Seung Hyoung Kim, and Sungwon Kim. “Self-supervised denoising of projection data for low-dose cone-beam CT”. In: *Med Phys* 50.10 (Apr. 2023), pp. 6319–6333.

[171] Dufan Wu, Kuang Gong, Kyungsang Kim, and Quanzheng Li. “Consensus Neural Network for Medical Imaging Denoising with Only Noisy Training Samples”. In: *arXiv preprint arXiv:1906.03639* (2019).

[172] Kihwan Choi, Joon Seok Lim, and Sungwon Kim. “Self-supervised inter- and intra-slice correlation learning for low-dose CT image restoration without ground truth”. In: *Expert Systems with Applications* 209 (2022), p. 118072. ISSN: 0957-4174.

[173] Dongdong Chen, Julián Tachella, and Mike E. Davies. “Equivariant Imaging: Learning Beyond the Range Space”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 4379–4388.

[174] Hiroyuki Kudo, Taizo Suzuki, and Essam A Rashed. “Image reconstruction for sparse-view CT and interior CT-introduction to compressed sensing and differentiated backprojection”. In: *Quant. Imaging Med. Surg.* 3.3 (June 2013), pp. 147–161.

[175] Hirofumi Kobayashi, Ahmet Can Solak, Joshua Batson, and Loic A. Royer. *Image Deconvolution via Noise-Tolerant Self-Supervised Inversion*. 2020.

[176] Adriaan Graas, Evert Wagner, and Felix Lucka. *Fluidized-bed-phantom radiographs from three Caesium-Iodine DALSA Xineos-3131 detectors for empirical PRF estimation*. Dataset on Zenodo. 2025.

Samenvatting

Introductie en probleemstelling

In dit proefschrift is onderzocht hoe computertomografie sneller gemaakt kan worden t.b.v. industriële en wetenschappelijke dynamische toepassingen.

Over computertomografie Computertomografie (CT) is een beeldvormende techniek, waarin met behulp van X-rays—ookwel bekend als röntgenstraling—in objecten of het menselijke lichaam gekeken kan worden. Dit werkt door röntgenfoto's te maken vanuit verschillende hoeken van het object of lichaam. Hiervoor gebruikt men een opstelling met een stralingsbron en een X-ray detector, vergelijkbaar met een fotocamera. Op een enkele röntgenfoto, genomen vanuit één hoek, is te zien hoeveel straling er door het medium heen is gekomen, en dat geeft het contrast tussen bijvoorbeeld bot (dat veel straling opneemt) en weefsel (dat relatief weinig straling opneemt) weer. Omdat één foto onvoldoende informatie bevat om een 3D beeld te vormen, combineert CT de röntgenfoto's vanuit vaak honderden tot duizenden verschillende hoeken. Hiervoor dienen de stralingsbron en detector rondgedraaid te worden om het lichaam, of kan een object op een draaischijf geplaatst worden. Het aantal hoeken bepaalt onder meer de scherpte van het uiteindelijk gevormde 3D beeld.

De noodzaak voor snelheid De rekenduur of rekenkracht die nodig is om van een aantal röntgenfoto's naar een 3D beeld te gaan, de zogenaamde *CT reconstructie*, hangt af van de resolutie van de röntgenfoto's en het 3D beeld, alsmede de rekencapaciteiten van de computer, en de keuze van het algoritme. Het filtered-backprojection (FBP) algoritme is bijvoorbeeld zo snel dat er binnen één seconde een reconstructie berekend kan worden. Ook de implementatie doet er toe: tegenwoordig wordt de videokaart veelvuldig gebruikt voor CT, omdat deze met een enkele computerinstructie gelijktijdig berekeningen kan uitvoeren op een gedeelte van de data. Veel moderne X-ray toepassingen vereisen echter nog snellere algoritmes en implementaties voor computertomografie, onder andere voor:

- snel bewegende processen, waarbij elk tijdstip in het proces in feite een eigen reconstructieprobleem is;
- algoritmes die meer rekenkracht vereisen, bijvoorbeeld omdat deze moeten compenseren voor lage aantallen hoeken, of voor hoge ruis op de röntgenfoto's;
- toepassingen die CT onafgebroken gebruiken, bijvoorbeeld aan de lopende band voor de opsporing van defecten in producten, of voor de controle van de voedselveiligheid;

- technieken gebaseerd op kunstmatige intelligentie (AI), i.h.b. neurale netwerken, die eerst moeten leren van een groot aantal gereconstrueerde voorbeelden voordat ze toegepast kunnen worden.

Samenvatting van de hoofdstukken

Hoofdstuk 2 In dit hoofdstuk wordt een experimentele opstelling geïntroduceerd die bestaat uit drie stralingsbronnen en drie detectoren. Doordat deze opstelling niet draait is deze geschikter voor het waarnemen van zeer snel bewegende processen, zoals de zgn. wervelbedden die gebruikt worden in de chemische industrie. Een wervelbed bestaat uit gas en bewegende deeltjes en gedraagt zich bij benadering als een vloeistof. Bestaande technieken konden tot dusver alleen wervelbedden waarnemen in één doorsnede op een bepaalde hoogte, of via een tijdsgemiddelde. Het voordeel van deze techniek is dat het de morfologische veranderingen van gasbellen in wervelbedden zowel door de tijd als door de 3D ruimte kan volgen.

Hoofdstuk 3 Dit hoofdstuk introduceert een neuraal network dat om kan gaan met de veranderlijke experimentele data binnen *real-time* tomografie. Real-time tomografie staat het toe om "live" mee te kijken met een experiment. Normaal gesproken leren neurale netwerken uit voorafgaande experimenten, voordat ze gebruikt worden. Echter, in de experimenten die met real-time tomografie worden uitgevoerd, in synchrotrons (deeltjesversnellers), is er vaak geen tijd of geschikte data voorafgaand aan het experiment. De voorgestelde methode leert daarom gedurende het experiment.

Hoofdstuk 4 Dit hoofdstuk introduceert een software pakket genaamd *ASTRA KernelKit*, dat het gemakkelijker maakt om tomografische algoritmes aan te passen voor bijzondere omstandigheden. Bestaande tomografische algoritmes zijn vaak alleen maar optimaal efficiënt voor standaardproblemen, bijvoorbeeld als de data voldoet aan veelvoorkomende afmetingen. De nieuwe software, gebaseerd op de programmeertaal Python, maakt het gemakkelijk voor wetenschappelijke gebruikers om aangepaste algoritmes te creeëren en uit te proberen op videokaarten, en kan met behulp van *tuning* de snelste versie voor een voorgedefinieerd probleem en specifieke hardware vinden.

Hoofdstuk 5 In dit hoofdstuk wordt tenslotte een oplossing gezocht voor het verwijderen van ruis op röntgenfoto's, wat ook de kwaliteit van de daaropvolgende CT reconstructies ten goede komt. State-of-the-art methodes voor het verwijderen van ruis zijn tegenwoordig allemaal AI-gebaseerd, maar hebben in de regel voorbeelden nodig van afbeeldingen zonder ruis om van te kunnen leren. Voor X-ray opstellingen zijn deze in de praktijk niet verkrijgbaar, omdat een ruisloze afbeelding een lange meettijd nodig heeft. De gepresenteerde oplossing past de eigenschappen van de ruis in röntgenfoto's van moderne X-ray scintillator detectoren dusdanig aan dat elke pixel daarna statistisch onafhankelijk wordt. Deze statistische eigenschap maakt de data vervolgens geschikt voor de zgn. Blind-Spot Netwerken, een type neurale netwerken waarvoor geen ruisloze voorbeelden meer nodig zijn.

Summary

Introduction and problem statement

This dissertation studies how the efficiency of Computed Tomography can be improved for industrial and scientific dynamic applications.

About Computed Tomography Computed Tomography (CT) is an imaging technique that makes it possible to look inside objects or the human body, using X-rays. This is done through the acquisition of X-ray images, called *radiographs*, taken from multiple angles around the object or body. For this purpose, a set-up is used that consists of a radiation source and X-ray detector, similar to an optical camera. A single radiograph, taken from one angle, shows how much radiation has passed through the materials, yielding a contrast, for example between bone (which absorbs a high quantity of X-rays) and tissue (which has a relatively low quantity of absorption). Since one radiograph does not contain sufficient information to form a 3D image, CT combines radiographs from often hundreds to thousands of angles. To achieve this, the radiation source and detector need to be rotated around the body, or an object can be positioned on a rotation stage. The number of angles determines, among other factors, the sharpness of the resulting 3D image.

The need for efficiency The computation time or computational power required to go from a set of radiographs to a 3D image—the so-called *CT reconstruction*—depends on the pixel resolutions of the radiographs and 3D image, as well as on the computational capabilities and the choice of algorithm. The filtered-backprojection (FBP) algorithm, for instance, is efficient enough to compute a reconstruction within just one second. The hardware implementation matters, too. Nowadays, graphics cards are widely used for CT, because they can perform computations on a part of the data synchronously with a single computer instruction. However, many modern X-ray applications require even faster algorithms and implementations for Computed Tomography, including:

- rapidly moving processes, in which each moment in time is effectively its own reconstruction problem;
- algorithms that require more computational power, for example because they must compensate for a low number of angles or a high noise level in the radiographs;
- applications that use CT continuously, for example for detecting production defects on conveyor belts, or for monitoring food safety;

- techniques based on artificial intelligence (AI), in particular neural networks, which must first learn from a large number of reconstructed examples before they can be applied to unseen data.

Summary of the chapters

Chapter 2 This chapter introduces an experimental setup consisting of three radiation sources and three detectors. Since this setup does not need to rotate, it is better suited for fast-moving processes, such as the fluidized beds used in the chemical industry. A fluidized bed consists of a mixture of gas and a particulate material, and behaves by approximation similar to a liquid. Existing techniques were previously only able to observe fluidized beds in a single cross-section at a certain height, or via a time averaged quantity. The advantage of the new technique is that it can track the morphological changes of gas bubbles inside fluidized beds both through time and the entirety of the 3D space.

Chapter 3 This chapter introduces a neural network that can handle the variable experimental data that is often encountered in *real-time* tomography. Real-time tomography makes it possible to observe an experiment live. Normally, neural networks need to learn from prior experiments before they are used. However, in the real-time tomography experiments that are performed at synchrotrons (particle accelerators), there is often insufficient time or suitable data prior to the experiment. The proposed method therefore learns during the experiment itself.

Chapter 4 This chapter introduces a software package called *ASTRA KernelKit*, which makes it easier to adapt tomographic algorithms for atypical reconstruction contexts. Existing tomographic algorithms are often only optimally efficient for standard problems, for example when the measurement data conform to commonly-used dimensions. The new software, based on the Python programming language, makes it easier for scientific users to create customized algorithms, experiment with them on graphics cards, and the software can use *kernel tuning* to find the fastest version for a predefined problem and hardware context.

Chapter 5 Finally, this chapter seeks a solution for removing noise from X-ray radiographs, which also improves the quality of subsequent CT reconstructions. State-of-the-art methods for noise removal are nowadays all data-driven, but generally require data sets of of noise-free examples from which to learn. For X-ray setups, such examples are not available in practice, because a noise-free radiographs require long measurement times. The presented solution therefore modifies the noise characteristics of radiographs acquired from modern X-ray scintillator detectors in such a way that each pixel subsequently becomes statistically independent. This statistical property then makes the data suitable for Blind-Spot Networks, a type of neural network for which noise-free examples are no longer needed.

Publications

1. Adriaan B.M. Graas, Evert C. Wagner, Tristan van Leeuwen, J. Ruud van Ommen, K. Joost Batenburg, Felix Lucka, and Luis M. Portela. "X-ray tomography for fully-3D time-resolved reconstruction of bubbling fluidized beds". In: *Powder Technology* 434 (2024), p. 119269. ISSN: 0032-5910. DOI: 10.1016/j.powtec.2023.119269

2. Adriaan Graas, Sophia Bethany Coban, K. Joost Batenburg, and Felix Lucka. "Just-in-time deep learning for real-time X-ray Computed Tomography". In: *Scientific Reports* 13.1 (2023), p. 20070. ISSN: 2045-2322. DOI: 10.1038/s41598-023-46028-9

3. Adriaan Graas, Willem Jan Palenstijn, Ben van Werkhoven, and Felix Lucka. "ASTRA KernelKit: GPU-accelerated projectors for Computed Tomography using CuPy". In: *Applied Mathematics for Modern Challenges* 2.1 (2024), pp. 70–92. DOI: 10.3934/ammc.2024004

4. Adriaan Graas and Felix Lucka. "Scintillator decorrelation for self-supervised X-ray radiograph denoising". In: *Measurement Science and Technology* 36.6 (2025), p. 065415. DOI: 10.1088/1361-6501/addc06

5. Adriaan Graas, Evert Wagner, and Felix Lucka. *Fluidized-bed-phantom radiographs from three Caesium-Iodine DALSA Xineos-3131 detectors for empirical PRF estimation.* Dataset on Zenodo. 2025. DOI: 10.5281/zenodo.15383254

Curriculum Vitæ

Adriaan Graas was born in Hoorn, and brought up Woerden, the Netherlands. After working as a freelance software engineer in Bodegraven, he studied the BSc. Earth Sciences with a *beta+* track at Utrecht University, finishing in 2015. He completed the MSc. Mathematical Sciences at Utrecht University in 2018, with specialization in Scientific Computing and master thesis on *Dimensionality Reduction & Uncertainty Quantification in Seismic Waveform Inversion*. In the same year, he started his doctoral research at the Centrum Wiskunde & Informatica in Amsterdam, as part of the NDNS+ cluster on topic of Mathematics and Algorithms for 3D Imaging of Dynamic Processes, with Dr. Felix Lucka as co-promotor and daily supervisor, and Prof. Dr. Joost Batenburg as promotor. The research was embedded in the *Computational Imaging* group, currently headed by Prof. Dr. Tristan van Leeuwen. Adriaan lectured the master course *Laboratory Class Scientific Computing*, organised with Prof. Dr. Rob Bisseling. During his Ph.D. studies, he took courses in scientific writing and scientific conduct.