Real-time tomographic reconstruction

Buurlage, J.

Chapter 5

Real-time quasi-3D tomographic reconstruction

Tomography is an important non-destructive technique for studying the three-dimensional structure of samples in various scientific fields such as biology, material science, and medicine, as well as being broadly applied in industry. Increasingly, tomography is used to understand dynamic processes in detail, e.g., by imaging biological samples that vary with time [Moo+13], or by studying material properties in a changing environment [Pat+15; Gib+15].

The change from static to time-resolved tomography is accompanied by a steep increase in computational requirements for the tomographic reconstruction. Moreover, many experiments have controlled parameters that rely, e.g., on specific events happening in the sample, which can be hard to identify from projection images alone. This means not only that the reconstruction is computationally expensive, but also that the typical offline reconstruction does not fulfill current needs due to long computation times.

In addition to the need for real-time tomography, i.e., having access

---

This chapter is based on:

Real-time quasi-3D tomographic reconstruction. *JW Buurlage, H Kohr, WJ Palenstijn, KJ Batenburg*. Measurement Science and Technology 29 (6), 2018

to reconstructions while scanning, developments in acquisition hardware also contribute to the computational challenge. For instance, the number of pixels on detectors is growing, and the detectors are operating at increasing frame rates. Furthermore, real-time tomography scanners are being developed for, e.g., airport security setups [Tho+15; War+16], which are able to perform full scans in short time windows. This highlights the importance of efficient reconstruction techniques.

Current approaches to tackle the computational challenges in real-time tomographic reconstruction can be roughly subdivided into two groups. First, reconstruction algorithms that are computationally more efficient are being adopted. Two examples of this are the *gridrec* method [Dow+99; MS12] and methods based on the log-polar Radon transform [Nik+17]. Second, reconstruction algorithms can be run in parallel, either on distributed compute clusters or specialized hardware such as GPUs [PBS11; Pal+17; Xu+10]. However, while these approaches can lead to a dramatic reduction in reconstruction times, the computational demands for reconstructing the full 3D volume remain a bottleneck for truly real-time tomographic reconstruction. By realizing that while currently often full 3D reconstructions are made, the reconstructed volume is primarily viewed slice by slice, we observe that more computational work is done than necessary.

Instead, one can create a processing workflow where slices are only reconstructed on demand. In this way, the computational requirements can be reduced by orders of magnitude, and in many cases the required amount of data communication can also be significantly reduced. Filtered backprojection (FBP) type methods allow these slices to have an arbitrary orientation. From a user's point of view these slices can easily be shifted and rotated and effectively it is as though 3D data is available, while only a small number of slices are actually reconstructed at any time, as illustrated by Figure 5.1. With this shift in perspective, we make *quasi*-3D real-time tomographic reconstruction feasible, in the sense that the results are visually identical to an architecture where the full 3D volume is reconstructed and then viewed slice-by-slice, yet at a fraction of the computational cost.

In this chapter we present a new methodology for real-time reconstructions, together with a software stack implementing these ideas. In Section 5.1 we revisit the mathematical properties of FBP type methods, that enable us to reconstruct arbitrarily oriented slices without forming the full 3D volume. While these properties follow directly from the basic formu-

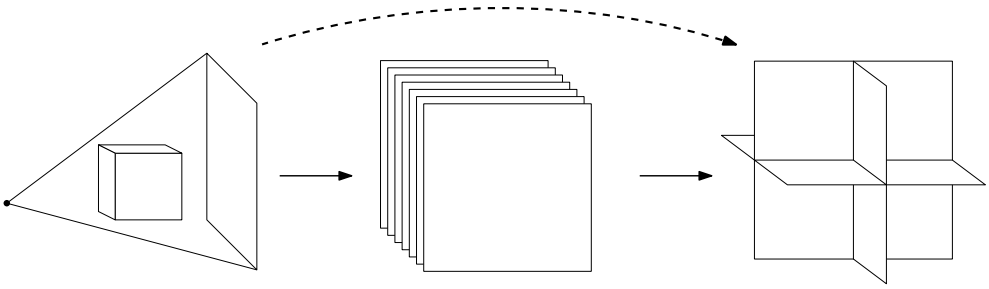

Figure 5.1: The solid arrows give a high-level overview of the data flow in a typical tomographic reconstruction setup. On the left, the projection data is acquired. In the middle, a reconstruction stack is created with an image for each slice along the rotation axis. From these slices, arbitrary slices of other orientations can be obtained through interpolation. In our new approach, represented with a dotted line, the generation of the reconstruction stack is skipped, and arbitrary slices are reconstructed directly from the projection data.

las, current approaches usually reconstruct the full 3D data at once. In Section 5.2, we present the interface and usage of the RECAST3D (REConstruction of Arbitrary Slices in Tomography) visualization software. It is a vital component of the proposed real-time reconstruction pipeline, as it allows the user to choose the slice(s) of interest in a dynamic way. In Section 5.3 we introduce the different components that are necessary to perform quasi-3D reconstructions. We highlight the unique distributed architecture of our novel reconstruction pipeline. Finally, in Section 5.4, we show that this new software greatly reduces reconstruction times, ultimately enabling almost instant slice reconstructions.

5.1 Reconstruction of arbitrary slices

Filtered backprojection (FBP) type methods for tomography are known to be very efficient in terms of numerical complexity and data usage. Whenever there are sufficiently many projections over the entire range of view angles, and the noise level is not too high, FBP typically performs very well also in terms of reconstruction quality. Here we understand as FBP any method that adheres to the “convolve, then backproject” workflow as shown in Fig-

ure 5.2. Examples of such methods are standard parallel beam FBP, the FDK algorithm for circular cone beam reconstruction [FDK84], and Katsevitch's algorithm for helical cone-beam reconstruction [Kat02] or general source trajectories [Kat03].

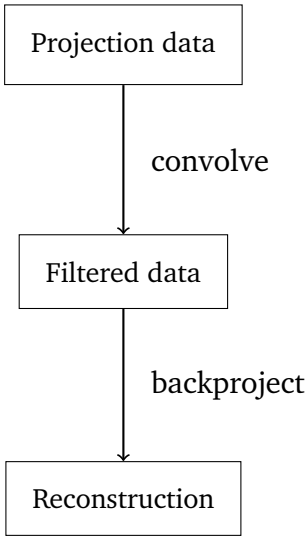

Figure 5.2: Workflow of filtered backprojection methods.

It is well-known that in 3D parallel beam geometry, horizontal slices can be reconstructed independently and from a single detector row. However, as we will demonstrate, FBP values in any subset of the reconstruction volume are mutually independent in any geometry. We start by recapitulating the well-known horizontal slice-by-slice reconstruction method in parallel beam geometry and then generalize to arbitrary slices and arbitrary geometries.

5.1.1 Parallel beam geometry

We consider the 3D parallel beam geometry with the z-axis as the only rotation axis (single-axis tilting). If f denotes a 3D volume, the corresponding projection data is given as the line integrals

$$g(\varphi, s, z) = \int_{-\infty}^{\infty} f\big(-t\sin\varphi + s\cos\varphi, t\cos\varphi + s\sin\varphi, z\big)\,\mathrm{d}t.$$

In an idealized setting, these values are available for $\varphi \in [0, \pi)$ and $(s,z) \in \mathbb{R}^2$. Filtered backprojection now consists of a one-dimensional filtering operation with a filter $k : \mathbb{R} \to \mathbb{R}$ in the s variable for each z, followed by backprojection:

$$g_{\text{filtered}}(\varphi,s,z) = \int_{-\infty}^{\infty} g(\varphi,s-u,z)\,k(u)\,\mathrm{d}u, \tag{5.1}$$

$$f_{\text{FBP}}(x,y,z) = \int_0^{\pi} g_{\text{filtered}}\big(\varphi, x\cos\varphi + y\sin\varphi, z\big)\,\mathrm{d}\varphi. \tag{5.2}$$

From (5.1) and (5.2) it is immediately clear that horizontal slices

$$f_{z_0}(x,y) = f(x,y,z_0),$$

with fixed $z = z_0$ can be reconstructed from a single data row $g_{z_0}(\varphi,s) = g(\varphi,s,z_0)$, i.e.,

$$f_{z_0,\,\text{FBP}}(x,y) = \int_0^{\pi} g_{z_0,\text{filtered}}\big(\varphi, x\cos\varphi + y\sin\varphi\big)\,\mathrm{d}\varphi.$$

In fact, if one is interested only in $f_{z_0}(x,y)$ for a single value z_0, then one has to perform also the filtering in (5.1) only for this fixed value of z_0. This reduces the whole task of reconstructing a horizontal slice to a two-dimensional problem.

Remark: It is important to notice that for all variants of single-slice reconstruction, the computed values in the slice are *identical* to the values in the full 3D reconstruction, restricted to the same slice.

The right-hand side of (5.2) refers only to the *current* reconstruction point (x,y,z), which implies that these points can be placed arbitrarily in 3D space. In particular, this mutual independence is not a special property of the parallel beam geometry but rather of the structure of the FBP algorithm itself. Hence it generalizes immediately to arbitrary slices and arbitrary geometries.

For instance, the remaining ortho-slices can be reconstructed as follows:

$$f_{x_0,\,\text{FBP}}(y,z) = \int_0^{\pi} g_{\text{filtered}}\big(\varphi, x_0\cos\varphi + y\sin\varphi, z\big)\,\mathrm{d}\varphi,$$

$$f_{y_0,\,\text{FBP}}(x,z) = \int_0^{\pi} g_{\text{filtered}}\big(\varphi, x\cos\varphi + y_0\sin\varphi, z\big)\,\mathrm{d}\varphi,$$

with the evident definitions

$$f_{x_0}(y,z) = f(x_0,y,z), \quad f_{y_0}(x,z) = f(x,y_0,z).$$

Again, the orthoslices contain the exact same values as a full volumetric reconstruction after restriction to these slices. Note, though, that both ortho-slices require the whole dataset since the z variable appears on both sides.

5.1.2 Cone beam geometry

We define the widely used *circular cone beam geometry* which is characterized by a point source moving on a circle of radius $r > 0$ in the x-y-plane and a flat detector on the opposite side of the same circle.

We parametrize the unit circle in the x-y-plane by

$$\boldsymbol{\theta}(\varphi) = (-\sin\varphi, \cos\varphi, 0), \quad \varphi \in [0, 2\pi).$$

Now we define the *source position* and the *detector piercing point* as two opposite points on a circle with radius $r > 0$ in the same plane:

$$\boldsymbol{a}(\varphi) = -r\boldsymbol{\theta}(\varphi), \quad \boldsymbol{p}(\varphi) = r\boldsymbol{\theta}(\varphi).$$

Finally we place a flat rectangular detector such that the ray from the source through the origin "pierces" the detector midpoint exactly at the piercing point $\boldsymbol{p}(\varphi)$, and orient the detector perpendicular to the piercing ray:

$$D(\varphi) = \left\{ \boldsymbol{p}(\varphi) + u\,\boldsymbol{\theta}^{\perp}(\varphi) + z\,\boldsymbol{e}_z \;\middle|\; -w/2 \le u \le w/2,\ -h/2 \le z \le h/2 \right\}.$$

Here, w and h stand for the width and the height of the detector, respectively, $\boldsymbol{\theta}^{\perp}(\varphi) = (\cos\varphi, \sin\varphi, 0) = -\boldsymbol{\theta}(\varphi + \pi/2)$ the unit vector tangent to the circle at angle φ, and $\boldsymbol{e}_z = (0,0,1)$. See Figure 5.3 for an illustration of the geometry.

This definition is straightforward to extend to arbitrary rotation axes and different radii for source and detector circles.

With these geometric conventions we define the projection data in circular cone beam geometry as

$$g(\varphi, \boldsymbol{y}) = \int_0^\infty f\big(\boldsymbol{a}(\varphi) + t\big(\boldsymbol{y} - \boldsymbol{a}(\varphi)\big)\big)\,\mathrm{d}t, \quad \varphi \in [0, 2\pi),\ \boldsymbol{y} \in D(\varphi). \tag{5.3}$$

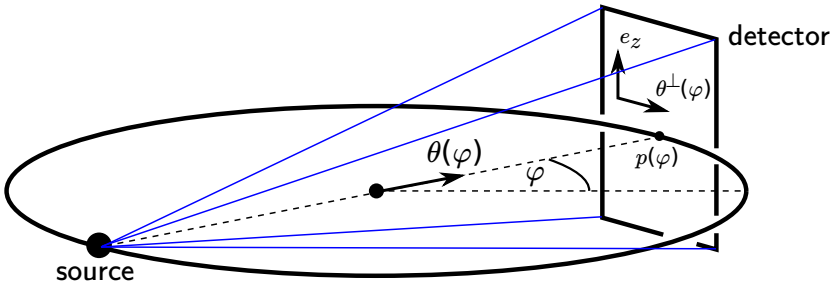

Figure 5.3: Sketch of a circular cone beam acquisition geometry as used by the backprojection (5.4).

It can be shown (see, e.g., [NW01]) that the backprojection for this geometry in a point $\boldsymbol{x} = (x, y, z)$ is

$$\mathrm{BP}[g](\boldsymbol{x}) = \int_0^{2\pi} \frac{1}{2r\,t(\boldsymbol{x},\varphi)^2}\, g\left(\varphi, \frac{\boldsymbol{x}\cdot\boldsymbol{\theta}^\perp(\varphi)}{t(\boldsymbol{x},\varphi)}, \frac{\boldsymbol{x}\cdot\boldsymbol{e}_z}{t(\boldsymbol{x},\varphi)}\right) \mathrm{d}\varphi, \tag{5.4}$$

where "$\cdot$" is the dot product in 3 dimensions and

$$t(\boldsymbol{x},\varphi) = \frac{\big(\boldsymbol{a}(\varphi) - \boldsymbol{x}\big)\cdot\boldsymbol{a}(\varphi)}{2r^2}$$

is the relative position of a reconstruction point $\boldsymbol{x} \in \mathbb{R}^3$ along the ray from the source point $\boldsymbol{a}(\varphi)$ to the detector through $\boldsymbol{x}$. Although the backprojection (5.4) is more involved to evaluate numerically, it still computes the value at a given volume point $\boldsymbol{x}$ independently from any other such point.

A very popular reconstruction method in circular cone beam geometry is the FDK algorithm [FDK84]. It consists of applying a one-dimensional filter k_{FDK} along the column coordinate u to preweighted measurements $\tilde{g}$, followed by the backprojection given in (5.4):

$$\begin{aligned}
\tilde{g}(\varphi, \boldsymbol{y}) &= \tfrac{\|\boldsymbol{p}(\varphi)-\boldsymbol{a}(\varphi)\|}{\|\boldsymbol{y}-\boldsymbol{a}(\varphi)\|}\, g(\varphi, \boldsymbol{y}),\\
g_{\mathrm{filtered}}(\varphi, u, z) &= \textstyle\int_{\mathbb{R}} \tilde{g}(\varphi, u-v, z)\, k_{\mathrm{FDK}}(v)\,\mathrm{d}v,\\
f_{\mathrm{FDK}}(\boldsymbol{x}) &= \mathrm{BP}[g_{\mathrm{filtered}}](\boldsymbol{x}).
\end{aligned}$$

Typically, k_{FDK} is chosen to be the *ramp filter*. To reconstruct an arbitrary slice $S = \boldsymbol{r} + \boldsymbol{n}^\perp$, $\boldsymbol{r}, \boldsymbol{n} \in \mathbb{R}^3$, $\boldsymbol{n} \neq (0,0,0)$, we can simply evaluate this

formula for all $\boldsymbol{x} \in S$. Just as for parallel beam reconstruction, the values computed in the slice are the same as if a full 3D FDK reconstruction was restricted to the same slice. In fact, the single-slice reconstruction avoids the interpolation step that would otherwise be incurred when restricting a full 3D reconstruction to a slice.

The FDK algorithm approximates the exact solution only in the central horizontal slice $z = 0$, while for other points in the volume, the data provided by circular cone beam acquisition is insufficient, leading to cone-beam artifacts. In [JKM11] the performance of FDK for experimental data is discussed. Certain extensions and modifications such as those that choose a specific filter, see, e.g., [Hah+13], also fit into our proposed framework.

To acquire complete data, one can additionally move both $\boldsymbol{a}$ and $\boldsymbol{p}$ with constant velocity $l/(2\pi)$ along the rotation axis $\boldsymbol{e}_z$ relative to the object, resulting in a helix instead of a circle:

$$\boldsymbol{a}(\varphi) = -r\boldsymbol{\theta}(\varphi) + \frac{l\varphi}{2\pi}\boldsymbol{e}_z, \quad \boldsymbol{p}(\varphi) = r\boldsymbol{\theta}(\varphi) + \frac{l\varphi}{2\pi}\boldsymbol{e}_z.$$

For this helical geometry, the reconstruction formula of Katsevich [Kat02] provides exact inversion. It is also of filtered backprojection type, even though both filtering and backprojection have more complex expressions. The formula induces a family of FBP methods by replacing the filter for exact inversion with a regularizing filter. In fact, for any piecewise smooth source trajectory satisfying certain geometric conditions, an exact FBP type reconstruction formula can be given [Kat03].

In conclusion, a method for the fast computation of a single-slice FBP reconstruction is useful for applications with either parallel beam or cone beam acquisition.

5.2 Software

Using the mathematical properties of FBP methods discussed in the previous section, we can introduce an optimized workflow for real-time visualization of tomographic reconstructions. In this section we present *RECAST3D*, visualization software that controls an on-demand reconstruction pipeline. In particular, it can be used for on-the-fly reconstruction of arbitrarily oriented slices. Our novel approach is to only compute a limited

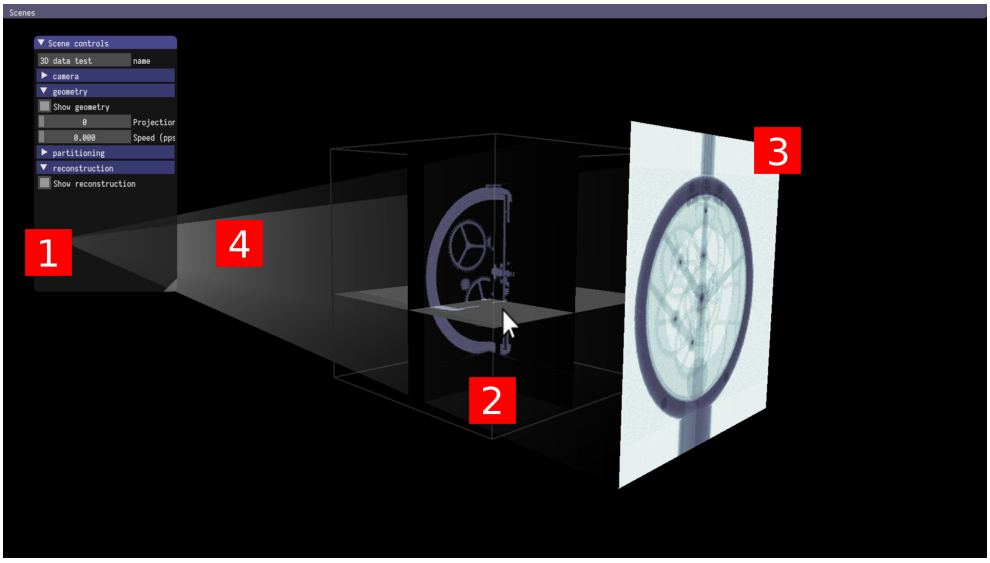

Figure 5.4: Screenshot of RECAST3D. Some simple analysis tools are provided in a GUI (1). In this example setup, three orthogonal slices are being shown in the middle (2) with the mouse currently hovering over one of them. A user can translate and rotate the planes by dragging them with the mouse. When the mouse button is released, the visualizer requests a reconstruction of the new slice. During the change of slice orientation and position, a low-resolution preview is shown. The interface is highly extensible. As an example we show the projection images (3) and the beam direction (4) in the same scene as the reconstruction, providing the user with additional information about the experimental setup.

number of slices, for example a set of three orthogonal slices, lowering the computational costs of the reconstruction tremendously. The slices that are being reconstructed can be changed with an intuitive interface. An exemplary screenshot of the visualization software is shown in Figure 5.4.

From a user's perspective, a typical workflow with RECAST3D is as follows. The tool is started on a workstation and connects to a reconstruction server that receives the relevant projection images. For small enough problems, this server can be the workstation itself. The software asks for specific slice reconstructions from the reconstruction server, initially presenting three orthogonal slices to the user. Assuming RECAST3D is used in a real-time setting, these are being reconstructed on-the-fly. The user can

hover the mouse over the slices and rotate and translate them in an intuitive manner. As new projection images arrive, the slices can be updated continuously.

We envision a modular system which we can extend gradually over time, as common needs and requirements become more clear. In the initial version of RECAST3D, next to the high resolution slices a low-resolution 3D preview is available when changing the orientation of a slice which allows the user to identify slices that are of particular interest. In addition, we show the projection images and visualize the acquisition geometry in the same 3D scene as the reconstruction. This presents the user with even more insight on the data that is coming in in real-time. It is possible to, e.g., change the color scheme that is used, or to rescale the data.

5.3 Implementation

The implementation of RECAST3D required a complete redesign of the typical tomographic reconstruction pipeline. In our discussion here we distinguish between three different stages of the reconstruction pipeline: acquisition, reconstruction and visualization. Note that in an actual experimental setting, we will need additional operations such as flatfielding and ring artefact correction. In the realization of our new quasi-3D reconstruction pipeline, all these stages work together with the common goal of giving the user a real-time quasi-3D reconstruction. To ensure the flexibility and scalability of our pipeline the system is completely distributed, in the sense that communication between the software components for the different stages happens through well-defined packets using a message passing protocol. The software stack consists of three main components:

1. *Reconstruction software* that is capable of performing the reconstruction of an arbitrarily oriented slice.

2. Definitions of the various *packets* supported by our communication protocol, together with a software library for constructing, sending, receiving, and parsing these packets.

3. The *software for real-time visualization*, RECAST3D, which is also the *control center* for the distributed software stack.

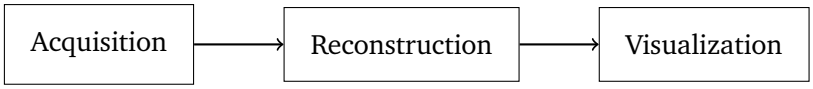

Figure 5.5: A simplified, but typical tomography pipeline, where the common pre- and post-processing steps are ignored. We emphasize here its linearity, i.e., data proceeds in its entirety from one stage to the next. Furthermore, in most cases these phases happen completely in a sequential manner.

Together, these components form an implementation of an extended reconstruction pipeline. Typically, the data in a tomography setup flows as in the linear pipeline shown in Figure 5.5. The software stack we introduce puts all the components in direct and real-time contact, enabling finer control over the dataflow, as shown in Figure 5.6. This has a number of advantages. We list some of them, in no particular order:

- Only subsets of the data have to be sent (or are requested) between the different stages.
- The computational requirements are significantly reduced, since only the slices that are shown are reconstructed.
- Since the entire system is integrated, the rich feedback allows the user to perform experiments faster and more efficiently

Distributed architecture

As mentioned in the previous section, our distributed architecture is based on a message passing protocol. Here, we describe in detail the different concepts and parts used in the distributed pipeline.

An experiment, or reconstruction, is captured in the system as a *scene*. These scenes consist of a number of data objects, such as reconstructed slices, projection data, and information on the acquisition geometry.

The central concept in the distributed pipeline is that of a *packet*. There are various packets that are used for communication, some examples are given in Listing 5.1. Every packet contains metadata used to identify an object in question (e.g., an identifier for a scene, and a slice), and perhaps

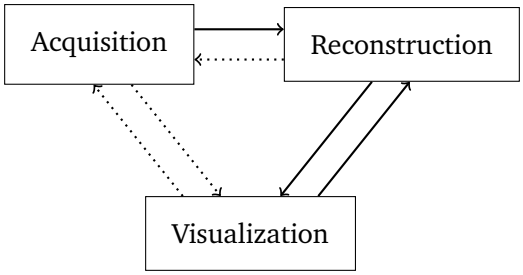

Figure 5.6: An extended complete pipeline, cf. Figure 5.5. All different stages are in direct contact, and no longer happen sequentially but in parallel. The implementations of the stages of the tomographic pipeline now communicate and coordinate with each other, reducing the dataflow and computational requirements. Although our distributed pipeline supports all communication paths, only the solid arrows are currently used.

some payload (i.e., a projection image, or a reconstructed slice) together with fields describing the payload such as the number of pixels or the position of the detector and source.

The packets that are described are independent of the specific technology used for sending them. In our reference implementation, ZeroMQ streams are used for communication. The core of the software stack is written in the C++ programming language.

Because the architecture is completely distributed, all components can be used independently and they are easily extensible. This modular approach allows users of our software to easily use or replace parts of the pipeline to suit their own purposes. Bindings to the Python programming language are provided, giving an accessible customization point. See also Listing 5.2 for an example of a custom script in our framework, which is able to completely replace the reconstruction component.

More generally, an important internal guideline for the development of this new pipeline is that it should be able to leverage existing and future software that is developed for image reconstruction. The library and specification take care of the necessary communication and coordination. The extended pipeline is implemented on a high level, rather than modifying existing software. Instead, existing software is used wherever possible. This gives our new system the great advantage of supporting custom software, from acquisition to reconstruction to visualization. Our current

```
struct GeometrySpecification {
    int32_t scene_id;
    bool parallel;
    int32_t projections;
    std::array<float, 3> volume_min_point;
    std::array<float, 3> volume_max_point;
};

struct SliceData {
    int32_t scene_id;
    int32_t slice_id;
    std::array<int32_t, 2> slice_size;
    std::vector<uint32_t> data;
};
```

Listing 5.1: Example packets, represented as a record data structure in the C++ programming language. The first packet defines some global information on the acquisition geometry: the number of projections, whether it describes a parallel or cone beam setup, together with the object volume which describes a bounding box for the sample being imaged. The second packet defines the data for a specific slice, with fields for the number of pixels together with the raw reconstructed data.

reconstruction server is built on top of ODL (the Operator Discretization Library [AKÖ17]) for describing the required geometric transformations at a high level, and the ASTRA Toolbox [Aar+16] for GPU-accelerated back-projection, customized for single slice processing.

Our software is available in open-source repositories, and can be found at `https://github.com/cicwi/`.

5.4 Results

In this section we compare the computational performance (i.e., the speed of reconstruction) of quasi-3D reconstructions to full 3D reconstructions. For the results presented here, the reconstructions are performed on a single node. This node has two Intel Xeon E5-2623v3 processors, 128 GB

```
import tomop

def reconstruction_callback(slice_geometry):
    data = custom_slice_data_function(slice_geometry)
    return data

server = tomop.server("Scene title", "tcp://localhost:5555")
server.set_callback(reconstruction_callback)
server.serve()
```

Listing 5.2: Example script for custom on-demand slice reconstruction. When the user rotates, translates, or creates a slice in the visualization interface, the system will request the new data for this slice using the user-supplied callback function. In the first line, the tomopackets library is imported. Next, a callback function is defined that takes an orientation, reconstructs the corresponding slice, and returns that reconstructed data. Below that, it is shown how to setup and connect a server.

RAM, and two dual-GPU NVIDIA GTX TITAN Z cards for a total of 4 GPUs with 6GB RAM each. The projection data has been prerecorded and pre-filtered, and is directly available to the reconstruction software. During a scan, the filtering can be done at the detector while taking images, without impacting the reconstruction time.

We use simulated data in our experiments. The test geometry is a circular cone beam geometry with rotation axis $\mathbf{z}$. The object has size $N \times N \times M$. The virtual detector is of size $N \times M$ and is positioned at the origin. The source is at distance $10 \times N$ from the center of the object. We take a total of N projections. Here, N and M are varied throughout our experiments.

The number of detector pixels that are required for the reconstruction of a single slice depends on the orientation of the slice (see also Section 5.1). We consider three slices: 1. an *axial slice* is a slice orthogonal to the rotation axis, 2. a *vertical slice* is parallel to the rotation axis, 3. a slice in between these extremes is a *tilted slice*.

We compare the timings of a full 3D reconstruction, with the timings of slice-based reconstructions for various orientations in Table 5.1. Some examples of the reconstructed slices are shown in Figure 5.7. Note that,

voxels	GPUs	full 3D	axial	vertical	tilted
$256 \times 256 \times 256$	1×	0.84 s	26.5 ms	22.6 ms	23.8 ms
	4×	0.31 s	35.9 ms	26.6 ms	22.9 ms
$512 \times 512 \times 512$	1×	1.07 s	33.4 ms	22.6 ms	31.8 ms
	4×	0.60 s	40.4 ms	27.2 ms	23.5 ms
$1024 \times 1024 \times 1024$	1×	17.3 s	61.6 ms	64.8 ms	63.1 ms
	4×	6.69 s	38.5 ms	39.1 ms	37.2 ms
$2048 \times 2048 \times 1024$	1×	274 s	286 ms	5.22 s	5.48 s
	4×	65.0 s	100 ms	106 ms	105 ms

Table 5.1: Reconstruction times for full 3D data, compared to reconstruction times for 2D slices of various orientations. See the text for a description of the hardware and test geometry. Here, the axial and vertical slices are taken at the center of the volume. The tilted slice is an axial slice, rotated 45° around the $\mathbf{x}$ axis. We consider a varying number of reconstructed voxels, corresponding to the $N \times N \times M$ volumes in the text. The performance when using a single GPU or multiple GPUs is also compared. For the relatively low numbers presented here, the standard deviation can be as high as 20% of the measurement, while for higher resolutions the numbers get relatively more stable with standard deviations of about 10% of the measurement.

as explained in Section 5.1, the single slice reconstructions are identical to reconstructions that would be obtained from a full 3D reconstruction. In particular, there is no loss of accuracy. The results show that individual slices can be computed quickly, even at high resolutions. The distributed system induces some overhead, which is included in the numbers presented. These can be a significant part of the total reconstruction times, particularly at lower resolutions. Using multiple GPUs can significantly decrease the reconstruction times, especially at high resolutions. For the highest resolution considered, the required data for reconstructing non-axial slices no longer fits on a single GPU which means that using multiple GPUs is a necessity for obtaining low reconstruction times.

When reconstructing vertical slices, already the complete data has to be filtered. In addition, the majority of the data is required for a backprojection. If all three orthoslices are required, then the complete data set is needed for the backprojection. However, the computational cost of the reconstruction always remains low. Because we visualize only individual

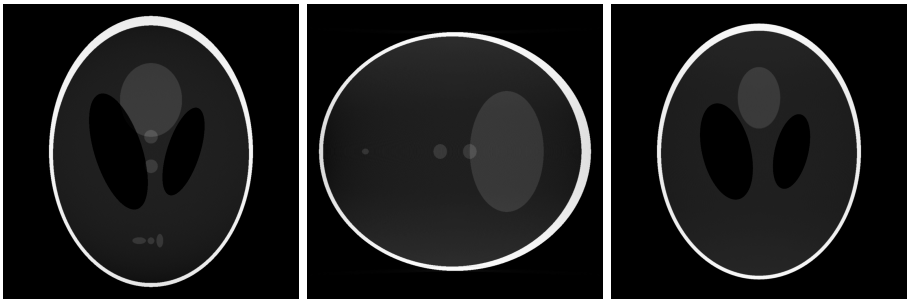

Figure 5.7: Reconstructed slices for a volume of $1024 \times 1024 \times 1024$ voxels. Here we used a modified 3D Shepp-Logan phantom. The left, middle and right reconstructed slices correspond to the axial, vertical and tilted slices as defined in Table 5.1.

slices, the amount of data required for visualization is always limited.

In our experiments we considered a circular cone beam geometry because in general it is a harder geometry to reconstruct than a parallel geometry. However, for quasi-3D reconstructions many properties that usually make reconstructing parallel geometries much simpler are lost, because slices of arbitrary orientation have to be reconstructed. In our experiments, we have observed similar performance for parallel geometries as for cone beam geometries.

5.5 Use cases

The ability to observe the internal state of the object in quasi-3D through the RECAST3D software is mainly valuable if real-time actions can be taken as a result of the observations, which would not be possible if one has to wait for a full 3D volume to be reconstructed. The RECAST3D software has several use cases, all related to various dynamic aspects of the image acquisition:

- **Dynamic processes within the object** of interest itself can be followed in real-time in a quasi-3D setting. For example, a bubble that moves through a liquid can be tracked by using three slices positioned in the center of the bubble and adjusting the slices to the observed direction.

- **Dynamic external parameters** related to the object state (temperature control, pressure control) can be adjusted to the observed state of the object. For instance, using a temperature controlled stage, the temperature of the object can be lowered until certain phase transitions occur inside the object (observed in the slices), after which the object is scanned at constant temperature.

- **Dynamic acquisition parameters** (source and detector positioning, rotation of the object) can be adjusted to the observed features of the object. For instance, the scanning geometry can be adjusted for the presence of metal (leading to artefacts) that has been observed at certain locations in the object and the object can be positioned closer to the source, zooming into a region-of-interest.

Moreover, the ability to quickly visualize several slices through the interior of the object while the object is in the scanner provides immediate feedback about the quality of the data, showing for example if the scan is good enough to resolve features of interest that are oriented in a particular direction chosen by the user.

5.6 Experiments

In this section we give two concrete examples of applications for the RECAST3D methodology.

The two datasets are acquired using the custom built and highly flexible FleX-Ray CT scanner, developed by XRE NV and located at CWI. The apparatus consists of a cone-beam microfocus X-ray point source that projects polychromatic X-rays onto a 1943×1535 pixels, 14-bit, flat detector panel. The acquired data is binned on the fly by 2-by-2 pixel windows, i.e., each raw projection is of size 972×768. The data is collected over 360 degrees in circular and continuous motion with 1200 projections distributed evenly over the full circle. For dataset A, the exposure time was 160 ms, the X-ray tube settings were 50kV, 50W, and we consider a limited detector window of size 1943×1135. For dataset B, exposure time was 100 ms, the X-ray tube settings were 40kV, 20W. The data is openly available online [CBB18].

As a first application, we give an example of a dynamic imaging situation where slice-based reconstruction can be sufficient. Consider a biomedical application where a needle is inserted into a subject or sample along a straight line, until some target is reached. First, the needle has to be located which can always be done by looking at, e.g., the standard three ortho-slices. After this, a slice containing the needle can be reconstructed dynamically. If necessary, this slice can be adjusted if the needle moves. To create a simplified test case for this use case, a needle-shaped structure was made out of Play-Doh and inserted in a box filled with poppy seeds (dataset A). As illustrated in Figure 5.8, a single projection is not sufficient to locate the needle, although the needle is visible. However, using the quasi-3D reconstruction a slice containing the needle can easily be identified.

As a second application, we consider an adaptive experiment where some finer structure is first located, after which a more detailed scan of this structure is made. An example would be to image growth rings in wood structures. This can be used, e.g., for non-destructive dendrochronology in archeological samples [Bil+12]. In the overview scan, the plane in which the growth rings lie can be found using our proposed methodology. After identifying this region, a high-resolution scan of this region can be made. As a test case we consider a piece of wood shaped as an egg (dataset B). In Figure 5.8, we show a single projection of the wooden egg, a quasi-3D visualization, and a slice containing the growth rings. Observe that in general it is hard to identify the growth-ring orientation from projection images alone.

5.7 Outlook and conclusions

In this chapter, we have introduced a new methodology for real-time quasi-3D tomographic reconstruction, and software implementing these ideas called RECAST3D. We show that reconstructing a limited number of arbitrarily oriented slices can be done at a fraction of the computational cost of a full 3D reconstruction, yet yielding similar information and insights for certain use cases.

In this work we focused on FBP and related reconstruction methods. In comparison, algebraic reconstruction methods lack the important proper-

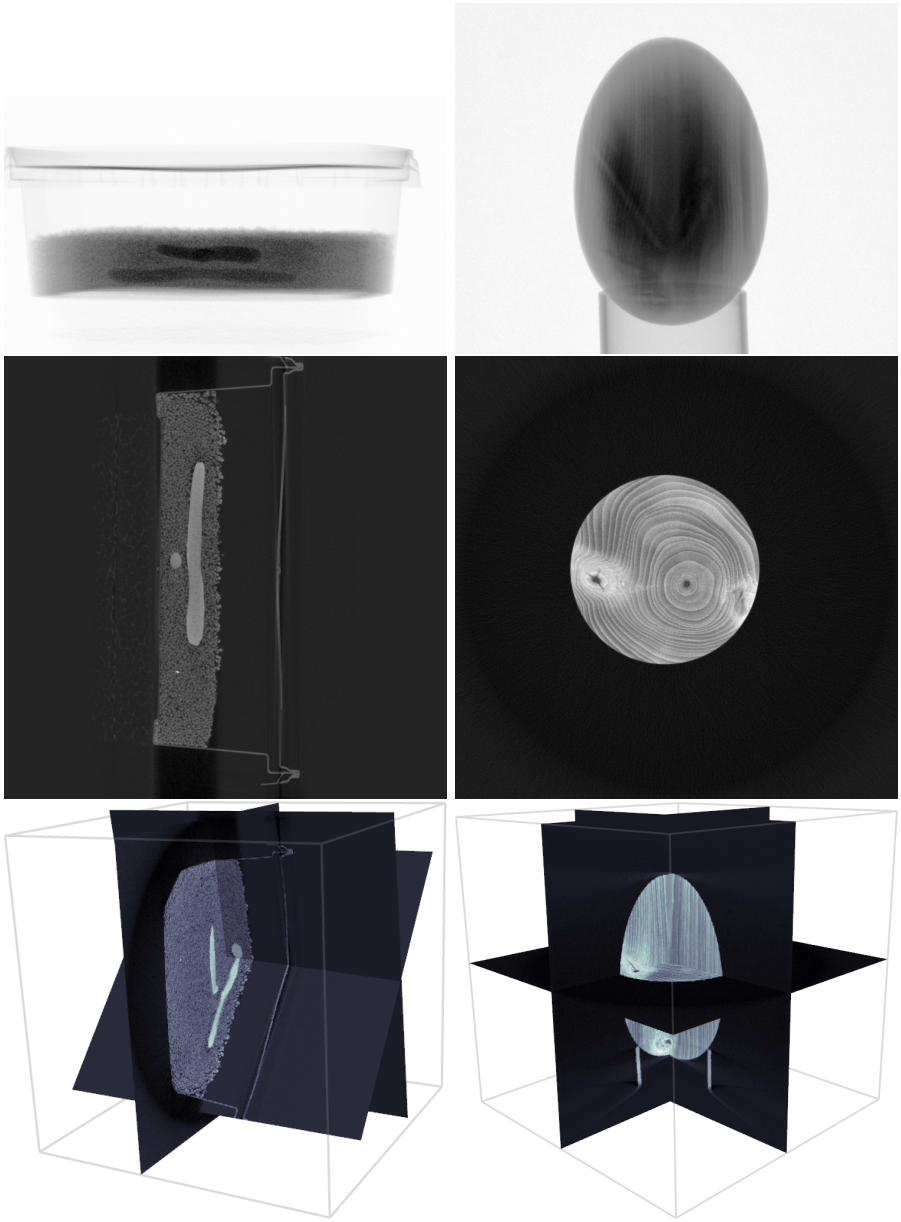

Figure 5.8: We show projections (top row), reconstructed slices (middle row) and quasi-3D reconstructions (bottom row). The contrast of the projections has been tuned by hand. On the left, dataset A is shown. On the right, dataset B is shown.

ties that we exploit. However, hybrid methods are conceivable which are tightly related to techniques for region-of-interest tomography. We expect that these more advanced reconstruction techniques can also fit into the framework presented here.

In addition to time-resolved experiments becoming more common, an interesting challenge will be to develop adaptive techniques. With these techniques, the scanning process itself can be steered based on the real-time reconstructions. Our distributed pipeline was developed specifically with this use-case in mind. Indeed, the cross-links between the different stages give rise to many interesting new possibilities. For example, the reconstruction cluster is able to control the scanner. This allows for algorithmically controlled experiments, that are driven dynamically by the reconstructions.