Plug-and-Play Learned Proximal Trajectory for 3D Sparse-View X-Ray Computed Tomography

Supplementary Material

In this supplementary material, we provide additional details on different aspects of the main text:

- In Sec. A, we present our private Cork-CBCT dataset as well as the public Walnut-CBCT dataset [21].
- In Appendix B, we provide the full mathematical development of the convergence analysis in Sec. 3.5.
- In Appendix C, we give more details on how we save the pre-defined optimization trajectory that we sample from during training.
- In Appendix D, we provide additional details on how we implement the evaluation, the training and inference procedures of our method, as well as compared approaches in Tabs. 1 and 2.
- In Appendix E, we detail how we perform the ablation study in Sec. 4.4.
- In Appendix F, we provide additional visual illustrations of the results presented in Sec. 4.

A Datasets

A.1 Cork-CBCT

The **Cork-CBCT** dataset we use in this work is a collection of 38 natural cork stoppers acquired with a Cone-Beam geometry acquisition setup. This dataset is composed of acquisitions with 720 radiographs, acquired with a 1024×1024 flat panel detector (Perkin Elmer XRD0821, CsI scintillator) along a circular trajectory with a 0.5 degree angular step. The acquisitions were performed with an X-ray tube (Viscom 225 kV micro-focus tube with a tungsten target) using the following parameters: intensity 450μA, voltage 40kV and no additional filter. The corresponding CT reconstructions of size $1024 \times 1024 \times 1024$ voxels are obtained using a primal-dual splitting algorithm [11]. The distance between the source and the rotation axis (`source-to-axis` distance), the distance between the source and the detector (`source-to-detector` distance) and the detector pixel size (`pixel-size`) are respectively 190 mm, 637 mm, and 0.2 mm. As emphasized in the illustrations in Fig. 6, the objects almost span the entire height of the detector but are not very large. For fair comparisons, the previously mentioned metrics are computed on a central crop of size 512×512. Similar to Coban *et al*. [16], to avoid cone-beam artifacts and the evaluation of the metrics on empty slices, we remove the first and last 150 slices of each reconstruction from the evaluation procedure.

A.2 Walnut-CBCT

For completeness, we also present the **Walnut-CBCT** dataset, although all details are present in the following work [21]. This dataset is composed of 42 walnut acquisitions,

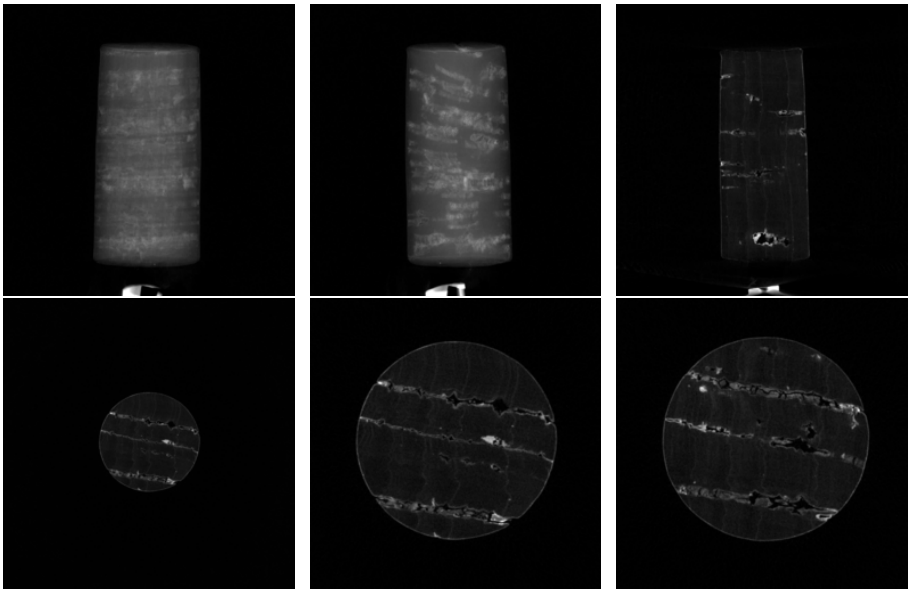


Fig. 6: Illustrations of the **Cork-CBCT** dataset. Top row: two examples of radiographs and one example of a sagittal reconstruction slice. Bottom row: examples of axial reconstruction slices, uncropped and cropped images.

with 1200 radiographs each, acquired with a 972×768 flat panel detector. Three different acquisitions are realized per sample to obtain sets of radiographs with different circular orbits. The corresponding ground truth reconstructions of size $501 \times 501 \times 501$ are obtained by solving a non-negativity constrained least-squares problem using 50 iterations of accelerated gradient descent. The projections of all circular orbits are used for the reconstruction to mitigate the cone angle artifacts due to the geometry of acquisition. During the evaluation, we use a central crop of size 300×300 on each slice, and we remove the first and last 100 slices of each reconstruction from the evaluation procedure.

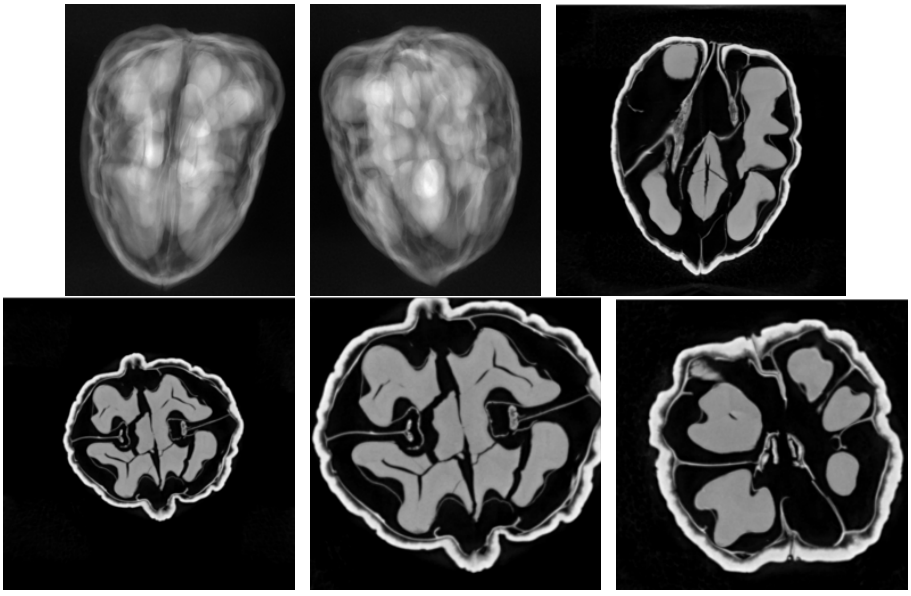


Fig. 7: Illustrations of the **Walnut-CBCT** dataset. Top row: two examples of radiographs and one example of a sagittal reconstruction slice. Bottom row: examples of axial reconstruction slices, uncropped and cropped images.

B Details on the convergence analysis Sec. 3.5

In our work, we define the problem to solve as

$$\hat{x} \in \underset{x \in \mathbb{R}^n}{\arg\min}\ \ell(x) + \lambda \mathcal{R}^*(x),\ \lambda > 0. \tag{18}$$

We show how to train a neural network D to approximate the proximity operator $\text{prox}_{\tau\lambda\mathcal{R}^*}$, with $\tau > 0$, the iteration step size. We then plug our Learned Proximal Trajectory operator into a PnP scheme

$$x_{k+1} = D_\alpha(x_k - \tau\nabla\ell),\ \alpha = \frac{\tau\lambda}{1+\tau\lambda}, \tag{19}$$

where D_α is the relaxed operator $D_\alpha(x) = \alpha D(x) + (1-\alpha)x$ and also the approximation of $\text{prox}_{\tau\lambda\mathcal{R}^*}$. Given that D is Lipschitz with Lipschitz constant $\beta > 0$, we show that D is a d-demicontractive operator with $d = 1 - \frac{2}{\beta+1}$ (Proposition 2). The convergence analysis of Cohen *et al*. [18, Theorem 4.5], on PnP with demicontractive operators applies, and we derive a condition on the regularization step $\gamma = \tau\lambda$ to obtain convergence, *i.e*. $\gamma\beta < 1$ (Theorem 1).

B.1 Lipschitz operator as demicontractive operator

In this section, we show that a β-Lipschitz mapping F is also d-demicontractive, and we show we estimate the constant d given β. As done in [18, Sec. 3.1 and Sec 3.2], we first review some concepts of fixed-point theory and useful definitions.

We start by considering a non-linear mapping $F : \mathbb{R}^n \to \mathbb{R}$, and we say that $x \in \mathbb{R}^n$ is a fixed point of F iff $F(x) = x$. We define the set of fixed points of F as

$$\text{Fix}(F) := \{x \in \mathbb{R}^n | T(x) = x\} \tag{20}$$

Definition 1 (Demicontractivity). *The mapping F is d-demicontractive with a constant $d < 1$ if for any $x \in \mathbb{R}^n$ and $z \in \text{Fix}(F)$ it holds that*

$$\|F(x) - z\|^2 \leq \|x - z\|^2 + d\|F(x) - x\|^2, \tag{21}$$

or equivalently

$$\frac{1-d}{2}\|x - F(x)\|^2 \leq \langle x - F(x), x - z\rangle. \tag{22}$$

Definition 2. *The mapping F is Lipschitz continuous with a constant $L > 0$ if for any $x, z \in \mathbb{R}^n$ it holds that*

$$\|F(x) - F(x)\| \leq L\|x - z\|. \tag{23}$$

When $L = 1$, F is said to be nonexpansive.

Definition 3 (Co-coercivity). *The mapping $r : \mathbb{R}^n \to \mathbb{R}$ is co-coercvie with a constant $\frac{1}{L} > 0$ if for any $x, z \in \mathbb{R}^n$ it holds that*

$$\frac{1}{L}\|r(x) - r(z)\|^2 \leq \langle r(x) - r(z), x - z\rangle \tag{24}$$

Proposition 1 (Co-coercivity of Lipschitz residual). *Assume a mapping $T(x) := r(x) - \alpha x$ is Lipschitz continuous with Lipschitz constant $\beta \leq \alpha$, then r is co-coercive with constant $\frac{1}{2\alpha}$.*

Proof. See [64, Proposition 2]

Proposition 2 (d-demicontractivity given a β-Lipschitz mapping). *Assume a mapping F is Lipschitz continuous with Lipschitz constant β, and that $\mathrm{Fix}(F) \neq \emptyset$, then F is also d-demicontractive with constant $d = 1 - \frac{2}{\beta+1}$.*

Proof. Given a mapping F, Lipschitz continuous with Lipchitz constant β, we define the residual $r(x) := x - F(x)$.The mapping r is also $(1+\beta)$-Lipschitz.

We define the mapping $T(x) := r(x) - \alpha x$. We see that for any $x, y \in \mathbb{R}^n$

$$\|T(x) - T(z)\| \leq (1 + \beta - \alpha)\|x - z\|$$

so T is $(1+\beta-\alpha)$-Lipschitz. For $\alpha = \frac{1+\beta}{2}$, T is α-Lipschitz and we know from Proposition 1 that r is co-coercive with modulus $\frac{1}{2\alpha} = \frac{1}{1+\beta}$.

The rest of the development follows from Cohen et al. *[18]. We assume that a fixed point of F exists and we see that the set $\mathrm{Fix}(F)$ is also the set of null points of r, $\mathrm{Null}(r) := \{x \in \mathbb{R}^n | r(x) = 0 \iff F(x) = x\}$.*
For any point $x \in \mathbb{R}^n$, for $z \in \mathrm{Null}(r)$ and r co-coercive with constant $(\frac{1}{1+\beta})$, we see that

$$\frac{1}{1+\beta}\|r(x)\|^2 \leq \langle r(x), x - z\rangle \tag{25}$$

Substituting r with $x - F(x)$ we obtain

$$\frac{1}{1+\beta}\|x - F(x)\|^2 \leq \langle x - F(x), x - z\rangle, \tag{26}$$

which is the definition of demicontractivity. Indeed, Eq. (26) *coincides with Definition 1, thus F is demicontractive with constant $d = 1 - \frac{2}{\beta+1}$.*

B.2 Convergence of LPT

We see that Eq. (19) coincides with [18, Eq. 4.6], thus we can easily derive a condition on the regularization step size γ. For completeness, we restate [18, Theorem 4.5].

Theorem 1. *Let D be a continuous d-demicontractive mapping and ℓ a proper convex lower semi-continuous differentiable function whose gradient $\nabla\ell$ is L-Lipschitz. Assume the following holds:*
(W1) $0 \leq d < 1$, $\alpha \in (0, \frac{1-d}{2}]$.
(W2) $\tau \in (0, \frac{2}{L})$.
(W3) $\mathrm{Fix}(D) \cap \mathrm{Fix}(G_\ell) = \emptyset$ *and* $\mathrm{Fix}(T) \neq \emptyset$ *where* $T(x) := D_\alpha(x - \tau\nabla\ell(x))$.
Then, the sequence $\{x_k\}_{k\in\mathbb{N}}$ generated Algorithm 1 converges to a fixed point of T.

Assuming that our learned operator D, in Sec. 4, is β-Lipschitz, we have shown in Proposition 2 that it is also d-demicontractive with $d = 1 - \frac{2}{\beta+1}$. Then it follows that condition (W1) of Theorem 1, with $\alpha = \frac{\gamma}{1+\gamma}$, is satisfied if $0 < \gamma\beta \leq 1$.

C Saving a pre-defined optimization trajectory

In this section, we give more details on how we save the pre-defined optimization trajectory (Fig. 2) that we sample from during training Sec. 3.4. Saving and storing a pre-defined optimization trajectory is very similar to the inference phase of our LPT scheme. The main difference is that we use knowledge of the *true* proximal operator $\mathrm{prox}_{\tau\lambda\mathcal{R}^*}$ in Eq. (11), rather than the learned approximation.

During the saving procedure, we sample x_0 and x^* from a dataset $\mathcal{D}$ and set an additional hyper-parameter $s > 0$ to control the number of 2D slices we extract per optimization step (x_k is 3D volume). The detailed procedure is given in Algorithm 2.

Algorithm 2 Saving a pre-defined optimization trajectory.

for $(x_0, x^*) \in \mathcal{D}$ **do**
  input: $x_0 \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $k = 0$, $K > 0$, step size $\tau > 0$, reg weight $\lambda > 0$, $\{q_k\}_{k \in \mathbb{N}}$.
  while $k < K$ **do**
    - $x_{k+\frac{1}{2}} = x_k - \tau \nabla \ell(x_k)$ ▷ gradient step on fidelity-term
    - $\texttt{extract}(x_{k+\frac{1}{2}})$ ▷ save s random 2D slices on the disk
    - $z_{k+1} = \mathrm{prox}_{\tau\lambda\mathcal{R}^*}(x_{k+\frac{1}{2}})$ ▷ true proximal step Eq. (11)
    - $x_{k+1} = z_{k+1} + q_k(z_{k+1} - z_k)$ ▷ inertial step
    - $k = k + 1$
  end while
end for

Disk space. The disk space required to save a pre-defined optimization trajectory is $\mathcal{O}(Kshw)$, with h and w the height and width of the 2D slices. On the Cork-CBCT dataset, if we consider 34 3D samples of size 1024^3 stored in `float32`, with $K = 200$ iterations of optimization, $s = 20$, $h = w = 1024$: the disk space required to save the intermediate reconstruction slices is ≈ 531 GigaBytes, this represents 136k data points. On the Walnut-CBCT dataset, using the same hyper-parameters s and K, the disk space required is 127 GigaBytes for 136k data points.

D Implementation details

We use `pyTorch` [45] to train the deep learning networks. To compute the forward and adjoint operator A and $A^\top$ for a Cone-Beam CT setup we use the `TIGRE` toolbox [9], which allows to split the operator computations into arbitrary smaller problems [10], and trades-off speed for size complexity.

D.1 Evaluation metrics

For each experiment, we evaluate the classic *Peak Signal to Noise Ratio* (PSNR) and the *Structural Similarity Index* (SSIM) [60] metrics. We measure the PSNR between an estimation $\hat{x}$ and the corresponding ground truth x^* as follows:

$$\mathrm{PSNR}(\hat{x}, x^*) = 10 \log_{10} \left(\frac{\mathrm{range}_x^2}{\mathrm{MSE}(\hat{x}, x^*)} \right), \tag{27}$$

where range_x is the difference between the maximum and minimum pixel value within a slice and MSE denotes the usual mean squared error between two images.

The SSIM is a metric based on human visual perception; it provides a quality index between 0 and 1 and is the aggregation of a sliding window computation at M different locations:

$$\mathrm{SSIM}(\hat{x}, x^*) = \\ \frac{1}{M} \sum_{j=1}^{M} \frac{(2\hat{\mu}_j \mu_j^* + C_1)(2\Sigma_j + C_2)}{(\hat{\mu}_j^2 + {\mu_j^*}^2 + C_1)(\hat{\sigma}_j^2 + {\sigma_j^*}^2 + C_2)} \tag{28}$$

where, $\hat{\mu}_j$ and $\hat{\sigma}_j$ are the mean and standard deviation of the image $\hat{f}$ at the j-th location, μ_j^* and σ_j^* are the mean and standard deviation of the image $f^\dagger$ at the j-th location. C_1 and C_2 are two constants used to stabilize the computation. We use the values $C_1 = (0.01\ \mathrm{range}_x)^2$ and $C_2 = (0.03\ \mathrm{range}_x)^2$. We re-use range_x to scale the SSIM computation. Following Wang *et al*. protocol in [60], we apply a Gaussian blur with a standard deviation of 1.5 to the images before computing.

The metrics are computed per slice as we found that the range of intensity values could vary a lot between different slices of the same object.

D.2 Learned Proximal Trajectory (LPT)

Before training. First, we need to save the pre-defined optimization trajectory from which we sample during training (Algorithm 2). The optimization trajectories are saved with λ_{train} while the inference is run with $\lambda_{\text{inference}}$. The Lipschitz constant L of the fidelity-gradient $\nabla \ell$ is computed using power iterations, as done in [52]. We report the Lipschitz constant L and the regularization parameters λ in Tab. 5.

Training. The training procedure is the one detailed in the main text Sec. 4. For more precision, we also use *Exponential Moving Average*, with decay 0.999, 0.5 of dropout, and clip the gradient norm to $1e-2$ to improve stability. Unless otherwise mentioned, we use these same improvement tricks to train the other methods.

Table 4: Regularization parameters and Lipschitz constant L of the fidelity-gradient $\nabla \ell$.

Learned Proximal Trajectory	Walnut-CBCT - **3D**			Cork-CBCT - **3D**		
Num. views	30	50	100	30	50	100
Lipschitz constant of $\nabla \ell$	569.2996	933.3025	1905.3225	116.64	192.6769	2781.5076
λ_{train}	3.5	5	9	0.75	1	2
$\lambda_{\text{inference}}$	7	10	18	0.75	2	2.5

Inference. The step size parameter is set to the same value $\tau = \frac{1}{L}$ during extraction (Algorithm 2) and inference (Algorithm 1). During inference, the regularization parameter λ can be adapted and set to a different value than the extraction procedure to obtain better results (Fig. 8).

Ultimately, the network D is only an approximation of $\text{prox}_{\gamma \mathcal{R}^*}$; hence, the optimization trajectory generated during inference is different from the one saved during training. Indeed, convergence is slower, thus during inference, we also set $K = 500$ iterations and set a stopping criterion $c_k = \|x_{k+1} - x_k\|_2 / \|x_0\|_2 < 10^{-4}$.

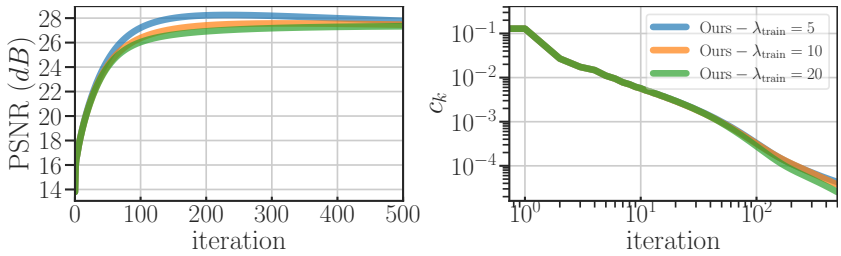


Fig. 8: In this figure, we show how changing the regularization weight λ during inference influences the convergence of the LPT scheme. The network D is trained on an pre-defined trajectory with λ_{train}, and the inference is run with $\lambda_{\text{inference}} = 10$. The best results are obtained with $\lambda_{\text{train}} = 5$, *i.e.* $\lambda_{\text{train}} < \lambda_{\text{inference}}$.

In practice, we obtain the best performance with D trained on finer optimization trajectories, *i.e.* a trajectory with small proximal step size $\gamma = \tau\lambda$. With a small step size γ, the procedure in Algorithm 2 converges slower to the solution x^*. However, training D on a finer optimization trajectory ensures that the network D sees more examples of the proximal operator during training, increasing its robustness.

D.3 PnP-PGD

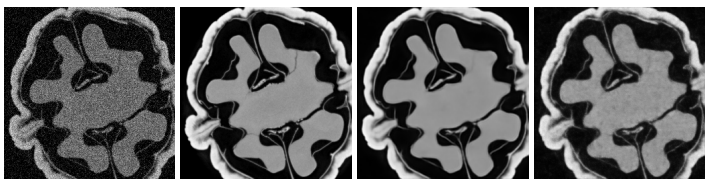
Training. We tried to train D using the procedure detailed in [46]. The denoising operator is defined as a learned maximally monotone operator, *i.e.* $D_\sigma(x) = \frac{x + Q_\sigma(x)}{2}$. We use the deep residual UNet architecture [49] in Fig. 11 (without the timestep embedding) for Q_σ, and concatenate the noise level σ to the input of the network. To

constrain the Lipschitz constant of Q_σ, we regularize the spectral norm of its Jacobian, thus minimizing the following loss function

$$\mathcal{L}(\theta) = \mathbb{E}_{x^*\sim\mathcal{D}}\mathbb{E}_{\xi_\sigma\sim\mathcal{N}(0,\sigma^2 I)}\Big[\|D_\sigma(x^* + \xi_\sigma;\theta) - x^*\|_2^2 + \mu\max(\|J_Q(\tilde{x})\|, 1-\varepsilon)\Big]. \quad (29)$$

First, D_σ is pre-trained for a regular Gaussian denoising task during 200k iterations, with Adam optimizer and cosine annealing with a learning rate of $1e-4$. During training σ is sampled from a uniform distribution $\mathcal{U}(0, i_{\max}\frac{10}{255})$, where $i_{\max}$ is the maximum voxel value of the dataset. We report $i_{\max} = 0.1179$ for the Cork-CBCT dataset and $i_{\max} = 0.502464$ for the Walnut-CBCT dataset. After pre-training, the Jacobian spectral norm of Q_σ is regularized during 100 epochs with Adam optimizer [35] and a learning rate of 10^{-4}, divided by 10 at epoch 80. We choose $\mu = 10^{-4}$ and $\varepsilon = 0.1$ for both datasets. The Jacobian's spectral norm $\|\cdot\|$ is estimated using 10 power iterations. We use a batch size of 32, and images are cropped to 256^2. We do not use *Exponential Moving Average* or dropout as it seems to interfere with the Jacobian spectral norm computation and induce significant differences between training and inference. We use the same parameters for both datasets.

Inference. Interestingly, similar to our observations on LPT in Sec. 4.4, we see that the Jacobian spectral norm regularization decreases the denoising performance of D_σ (Fig. 9). More importantly, when plugged into PnP-PGD (Eq. (5)), the constrained network makes the optimization unstable, and the algorithm diverges Fig. 10.



(a) Input **(b)** Ground-truth **(c)** Gaussian denoising network. **(d)** [Constrained] Gaussian denoising network.

Fig. 9: Comparison of denoising performance on the Walnut-CBCT dataset. The input **(a)** is the Gaussian contaminated ground-truth **(b)**. In **(c)**, the denoising operator is unconstrained. In **(d)**, the denoising operator is parametrized as a learned maximally monotone operator [46] and a regularized Jacobian spectral norm, with $\max(\|J_Q\|) = 0.9595$.

To circumvent this problem, we use the parametrization of [18], and relaxed the Gaussian denoising operator $D_\alpha(x) = \alpha D(x) + (1-\alpha)x$. We revert the network to a simple DRUnet, *i.e.* $D = D(\cdot;\theta)$ and train it for a Gaussian denoising task using the hyperparameters in Appendix D.2. Similar to our approach Eq. (19), we use $\alpha = \frac{\tau\lambda}{1+\tau\lambda}$.

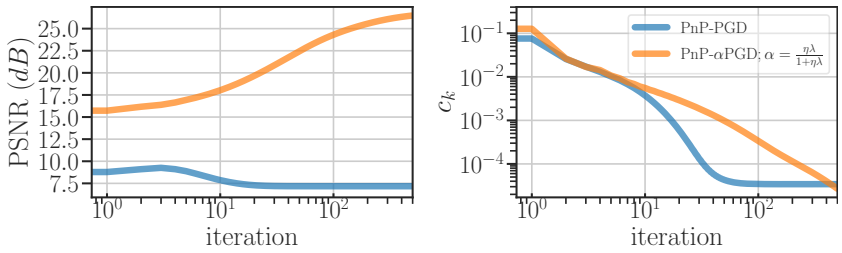


Fig. 10: Convergence of the classic PnP-PGD [46] and relaxed PnP-αPGD [18] for the sparse-view reconstruction task on the Walnut-CBCT dataset. In PnP-PGD, we regularize the Jacobian spectral norm of Q as in [46]. In PnP-αPGD, we use an unconstrained network D to build D_α.

Table 5: Regularization parameters for PnP-PGD.

PnP-PGD	Walnut-CBCT - **3D**			Cork-CBCT - **3D**		
Num. views	30	50	100	30	50	100
λ	8	10	20	2	4	4

D.4 InDI

The Inversion by Direction Iteration (InDI) algorithm [20] is a recent formulation for image restoration where a low-quality input y is gradually improved in small steps via a neural operator F. It is iterative by nature but does not require knowledge of the forward degradation process, as opposed to variational-based methods such as PnP. Given a clean and a low-quality image pair (x, y), they define a continuous forward degradation process by

$$x_t = (1-t)x + ty, \quad \text{with } t \in [0,1]. \tag{30}$$

They learn to reverse this process using a neural operator F that maps the current iterate x_t to the next iterate $x_{t-\delta}$, such that

$$x_{t-\delta} = \frac{\delta}{t}F(x_t, t) + (1 - \frac{\delta}{t})x_t, \quad \text{where } 0 \leq \delta \leq 1, \tag{31}$$

Similar to the optimization of D in Eq. (14), the neural operator F is train to reconstruct the ground truth x from any point x_t, that is

$$\min_\theta \mathbb{E}_{x,y\sim p(x,y)}\mathbb{E}_{t\sim p(t)}\|F(x_t, t; \theta) - x\|_p \tag{32}$$

We see with Eq. (31) that the next iterate $x_{t-\delta}$ is exactly the relaxed neural operator F_{α_t} with $\alpha_t = \dfrac{t}{\delta}$. Thus, InDI is very similar to the learned proximal operator of the squared Euclidean distance Eq. (11).

To compute the results in Tabs. 1 and 2, we trained InDI following the same procedure as [20]. We use the architecture in Fig. 11 for the network F, and choose $\delta = 0.01$,

i.e. we divide the iterative procedure into $T = 100$ steps. We use the same parameters for both datasets. The network is trained for 200k iterations with Adam optimizer and cosine annealing with a learning rate of $1e-4$. We use a batch size of 32, and images are cropped to 256^2. Interestingly, the network F is minimized with $p = 1$ in Eq. (32), so it minimizes an L^1 reconstruction error.

D.5 Architecture

For fair comparisons, we use the same architecture D for every experiment (Fig. 11). On both datasets, we use $c = 32$ channels, see Fig. 11, to maintain low computational cost due to our 3D experiments.

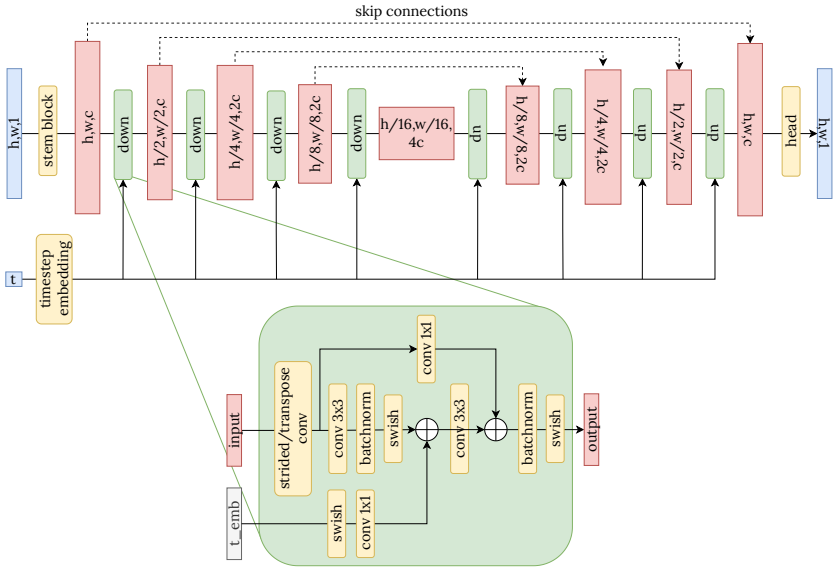


Fig. 11: Residual UNet architecture used for the restoration network D, and the other baselines in the experiments.

E Details on the ablation study Sec. 4.4

In Tab. 3, we emphasize how the optimization procedure of D, influences the performance once plugged in a PnP-PGD algorithm. We then show that the LPT procedure Eq. (13) is necessary to train a robust restoration prior.

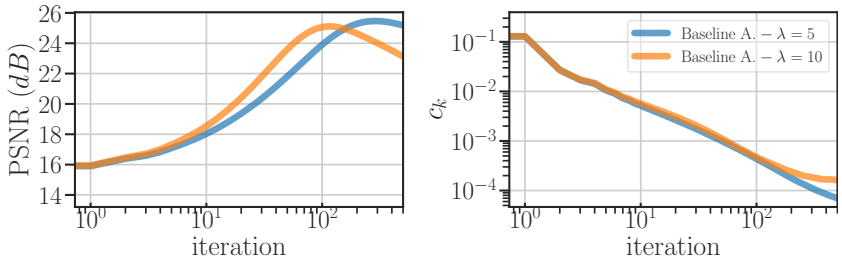


Fig. 12: In this figure, the network D from Baseline A. (Tab. 3 & Appendix E) is trained as a standard post-processing network (Eq. (10)) and thereafter plugged in an LPT inference scheme Algorithm 1. We show the influence of the regularization weight λ on the convergence, and the necessity to decrease λ to avoid divergence.

Baseline A. In this configuration, D is trained as a standard post-processing network (Eq. (10)), *i.e.* we only sample x_0 and not the intermediate reconstruction steps (Fig. 2). In this configuration, we decrease the regularization weight λ compared to the other configurations (Fig. 12), in order to avoid divergence during the inference procedure. Thus, we set $\lambda = 5$ (instead of $\lambda = 10$) for the Walnut-CBCT dataset and $\lambda = 1$ (instead of $\lambda = 2$) for the Cork-CBCT dataset.

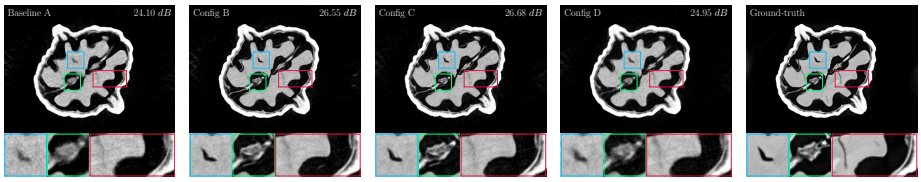


Fig. 13: Illustrations of sparse view (50/1200) reconstructions on the Walnut-CBCT [21] dataset using the configurations compared in Tab. 3. *Config C* is the configuration used in Tab. 1 in the main text.

Configuration B. & C. In configuration B, D is trained as a robust approximation of $\text{prox}_{\gamma\mathcal{R}^*}$ along a pre-defined optimization trajectory. In configuration C, D, we augment the input of D with iteration step conditioning. The training and inference procedures are the same as the results in Tabs. 1 and 2.

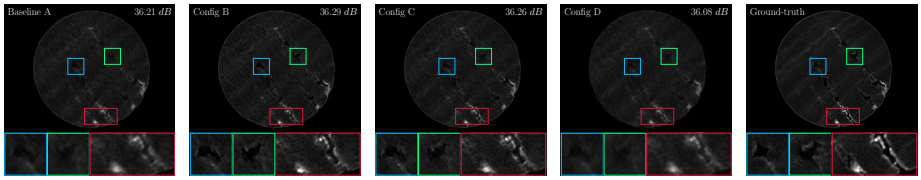


Fig. 14: Illustrations of sparse view (50/720) reconstructions on the Cork-CBCT dataset using the configurations compared in Tab. 3. *Config C* is the configuration used in Tabs. 1 and 2 in the main text.

Configuration D. The first part of the training procedure is the same as the results in Tabs. 1 and 2. Once pre-trained, we finetune D by regularizing its Jacobian spectral norm, as done in [46]. The Jacobian spectral norm is regularized during 100 epochs with Adam optimizer [35] and a learning rate of 10^{-4}, divided by 10 at epoch 80. We choose $\mu = 10^{-4}$ and $\varepsilon = 0.1$ for both datasets. The Jacobian's spectral norm $\|\cdot\|$ is estimated using 10 power iterations

F Additional illustrations of sparse view reconstructions

We also provide additional illustrations on the Walnut-CBCT and Cork-CBCT datasets.

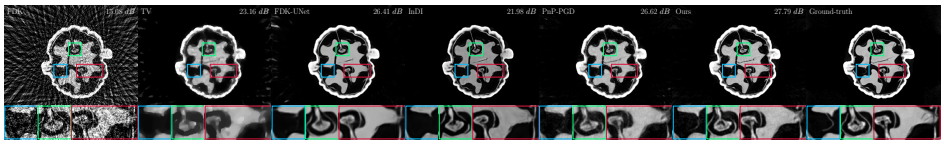


Fig. 15: Additional illustrations of sparse view (50/1200) reconstructions on the Walnut-CBCT [21] dataset using the methods compared in Tab. 1.

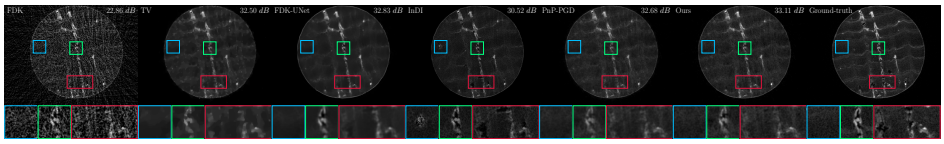


Fig. 16: Additional illustrations of sparse view (50/720) reconstructions on the Cork-CBCT dataset using the methods compared in Tab. 2.