# Sandwich time-of-flight proton CT image reconstruction using the ~~Adam~~ Adagrad optimizer with automatic differentiation

Aurélien Coussat[*,1] Jean Michel Létang[1] Nils Krah[2] Simon Rit[1]

[*]Presenting author
[1]CREATIS laboratory, Lyon, France
[2]HollandPTC, Delft, Netherlands

21/10/2025

# Outline

1. Introduction: sandwich time-of-flight pCT

2. Forward model and gradient descent reconstruction

3. Results

4. Conclusion

# Outline

1. Introduction: sandwich time-of-flight pCT
2. Forward model and gradient descent reconstruction
3. Results
4. Conclusion

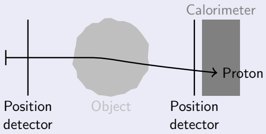
# Treatment planning of proton therapy

- Treatment planning for proton therapy requires the relative stopping power (RSP) distribution within the patient's body
- This distribution is currently obtained by converting Hounsfield units obtained from an X-ray computed tomography
- The conversion introduces inaccuracies in the RSP estimation[1], resulting in increased safety margins

[1] Ming Yang et al. In: *Physics in Medicine & Biology* 57.13 (2012)

# Energy-loss, TOF and sandwich TOF proton computed tomography

## Energy-loss pCT



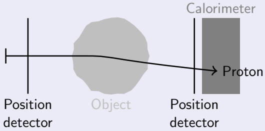
- Energy is detected using a calorimeter
- Challenging to have high aquisition rate

[2] William A. Worstell et al. In: *Medical Imaging 2019: Physics of Medical Imaging*. Vol. 10948. International Society for Optics and Photonics. 2019, Felix Ulrich-Pur et al. In: *Physics in Medicine & Biology* 67.9 (2022), Nils Krah et al. In: *Physics in Medicine & Biology* 67.16 (2022)

[3] F. Ulrich-Pur et al. In: *Journal of Instrumentation* 18.02 (2023)

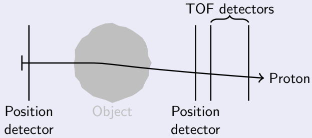
# Energy-loss, TOF and sandwich TOF proton computed tomography

## Energy-loss pCT



- Energy is detected using a calorimeter
- Challenging to have high aquisition rate

## TOF pCT[2]



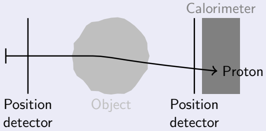
- Energy is estimated using TOF between two downstream detectors
- Placing TOF detectors with sufficient distance is impractical

[2] William A. Worstell et al. In: *Medical Imaging 2019: Physics of Medical Imaging*. Vol. 10948. International Society for Optics and Photonics. 2019, Felix Ulrich-Pur et al. In: *Physics in Medicine & Biology* 67.9 (2022), Nils Krah et al. In: *Physics in Medicine & Biology* 67.16 (2022)

[3] F. Ulrich-Pur et al. In: *Journal of Instrumentation* 18.02 (2023)

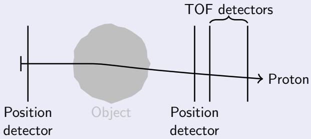
# Energy-loss, TOF and sandwich TOF proton computed tomography

## Energy-loss pCT



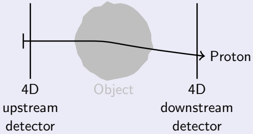
- Energy is detected using a calorimeter
- Challenging to have high aquisition rate

## TOF pCT[2]



- Energy is estimated using TOF between two downstream detectors
- Placing TOF detectors with sufficient distance is impractical

## STOF pCT[3]



- WEPL is directly estimated from the TOF between the detectors
- Can be implemented with two 4D LGADs

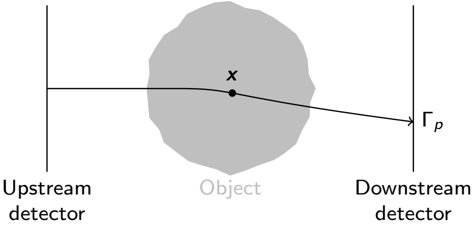
[2] William A. Worstell et al. In: *Medical Imaging 2019: Physics of Medical Imaging*. Vol. 10948. International Society for Optics and Photonics. 2019, Felix Ulrich-Pur et al. In: *Physics in Medicine & Biology* 67.9 (2022), Nils Krah et al. In: *Physics in Medicine & Biology* 67.16 (2022)

[3] F. Ulrich-Pur et al. In: *Journal of Instrumentation* 18.02 (2023)

# Mathematical model of STOF pCT

- TOF of proton $p$:

$$\tau_p = \int_{\Gamma_p} \frac{\mathrm{d}\boldsymbol{x}}{v_p(\boldsymbol{x})} \tag{1}$$



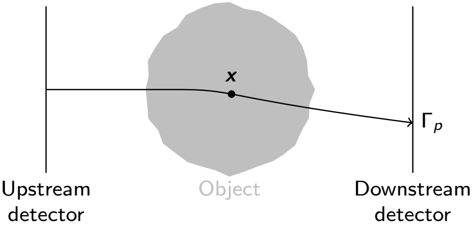
# Mathematical model of STOF pCT

- TOF of proton $p$:

$$\tau_p = \int_{\Gamma_p} \frac{\mathrm{d}\boldsymbol{x}}{v_p(\boldsymbol{x})} \qquad (1)$$

- Velocity of the proton $p$ at point $\boldsymbol{x}$:

$$v_p(\boldsymbol{x}) = c^2 \frac{E_p(\boldsymbol{x})}{E_p(\boldsymbol{x}) + mc^2} \sqrt{1 + 2\frac{mc^2}{E_p(\boldsymbol{x})}} \qquad (2)$$



# Mathematical model of STOF pCT

- TOF of proton $p$:

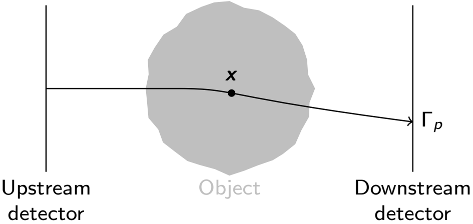
$$\tau_p = \int_{\Gamma_p} \frac{\mathrm{d}\boldsymbol{x}}{v_p(\boldsymbol{x})} \tag{1}$$

- Velocity of the proton $p$ at point $\boldsymbol{x}$:

$$v_p(\boldsymbol{x}) = c^2 \frac{E_p(\boldsymbol{x})}{E_p(\boldsymbol{x}) + mc^2} \sqrt{1 + 2\frac{mc^2}{E_p(\boldsymbol{x})}} \tag{2}$$

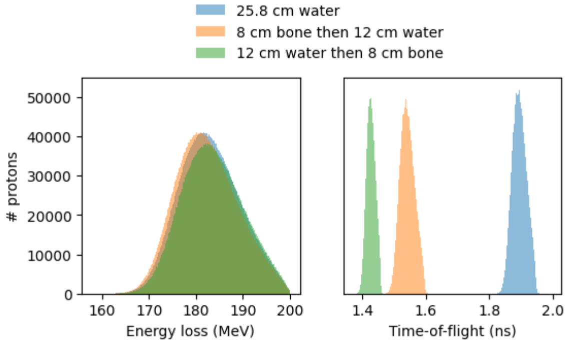
- Energy of the proton $p$ at point $\boldsymbol{x} = \Gamma_p(s)$:

$$E_p(\Gamma_p(s)) = E_{\text{in}} - \int_{-\infty}^{s} \underbrace{\rho(\Gamma_p(s'))}_{\text{RSP}} \underbrace{\sigma_{\mathrm{H_2O}}(E_p(\Gamma_p(s')))}_{\text{Stopping power of water}} \mathrm{d}s' \tag{3}$$



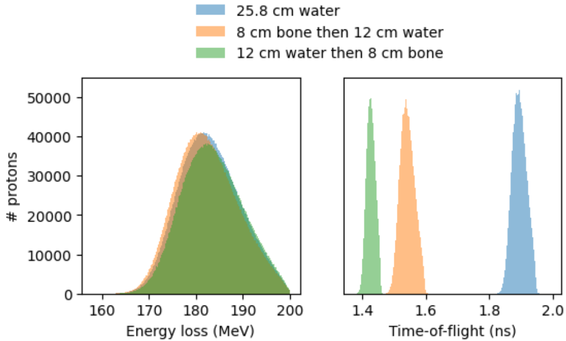
# Difficulty of sandwich time-of-flight pCT

- Protons traversing different materials with equivalent WEPLs result in
  - equal energy loss
  - different TOFs



# Difficulty of sandwich time-of-flight pCT

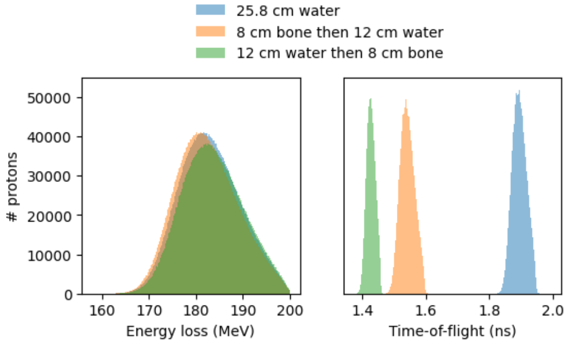
- Protons traversing different materials with equivalent WEPLs result in
  - equal energy loss
  - different TOFs



- The conversion between TOF and WEPL is non-bijective

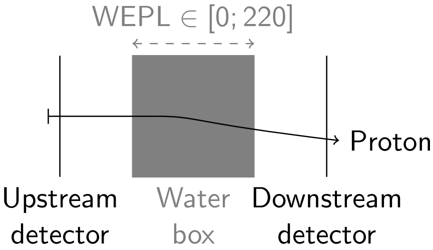
# Difficulty of sandwich time-of-flight pCT

- Protons traversing different materials with equivalent WEPLs result in
  - equal energy loss
  - different TOFs



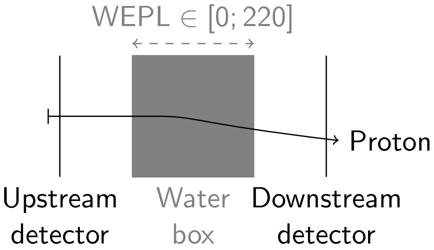
- The conversion between TOF and WEPL is non-bijective
- The forward problem is non-linear and non-trivial to invert

# First attempt using a 1D calibration curve



https://github.com/RTKConsortium/PCT/blob/main/applications/pctweplfit/pctweplfit.py

# First attempt using a 1D calibration curve



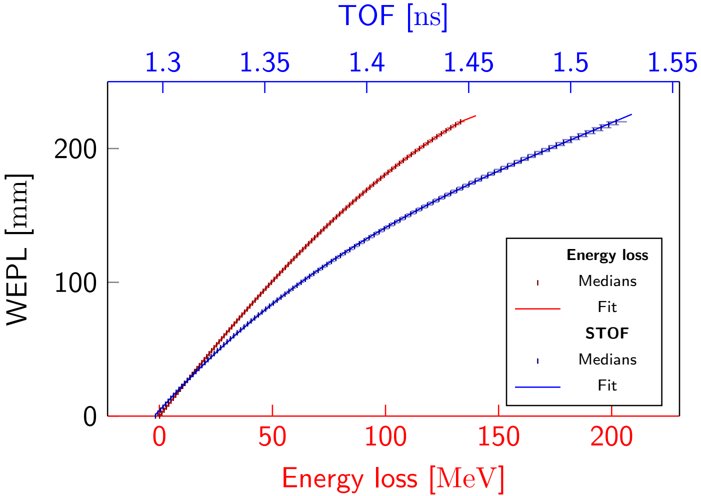
WEPL ∈ [0; 220]

Proton

Upstream detector

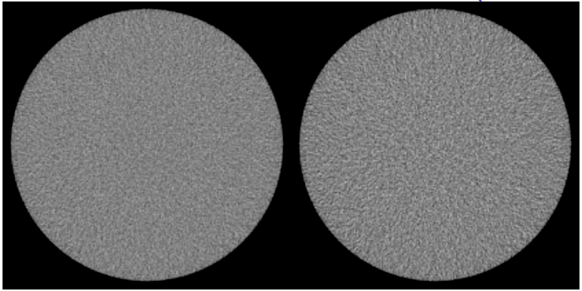
Water box

Downstream detector

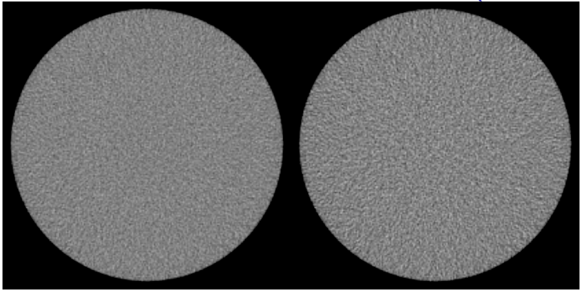


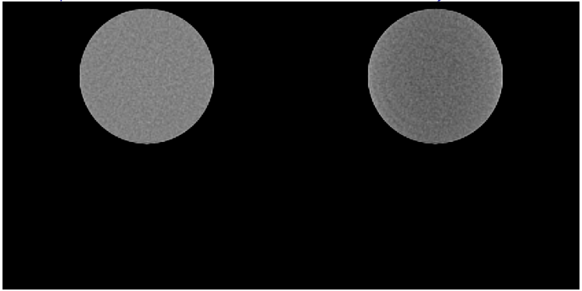
https://github.com/RTKConsortium/PCT/blob/main/applications/pctweplfit/pctweplfit.py

# Tomographic reconstructions (energy loss/sandwich time-of-flight)

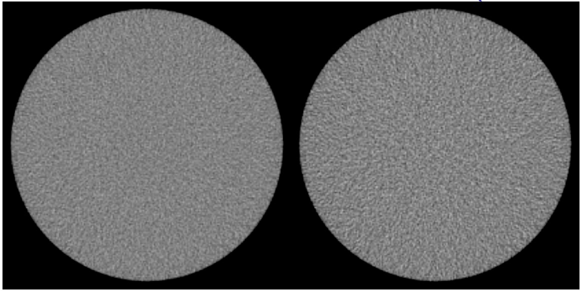


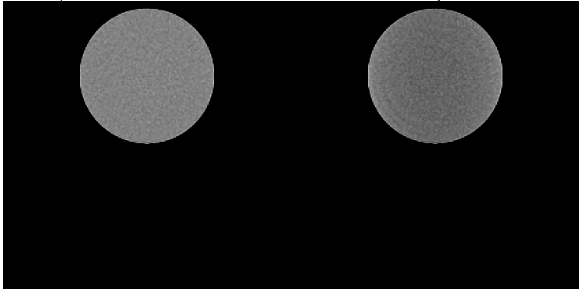
# Tomographic reconstructions (energy loss/sandwich time-of-flight)

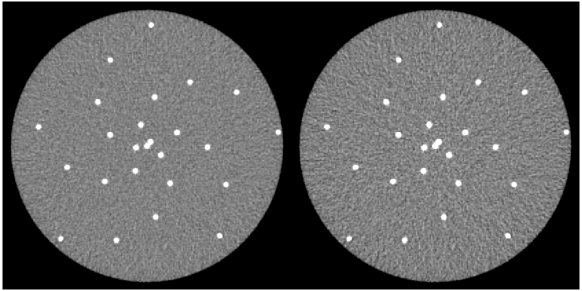




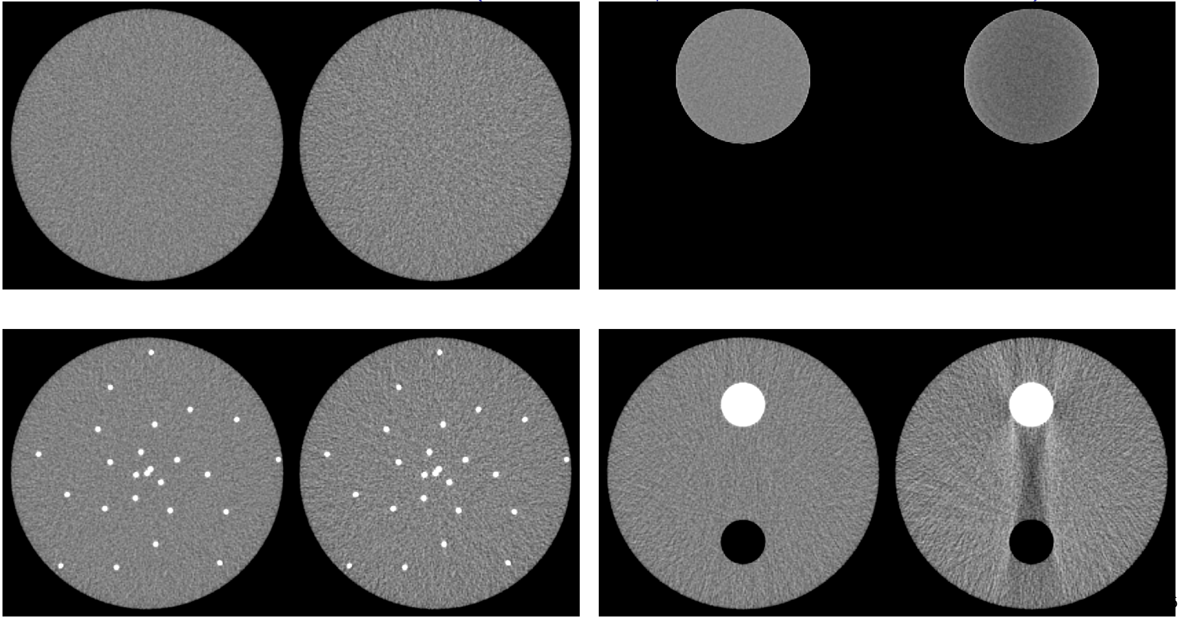
# Tomographic reconstructions (energy loss/sandwich time-of-flight)



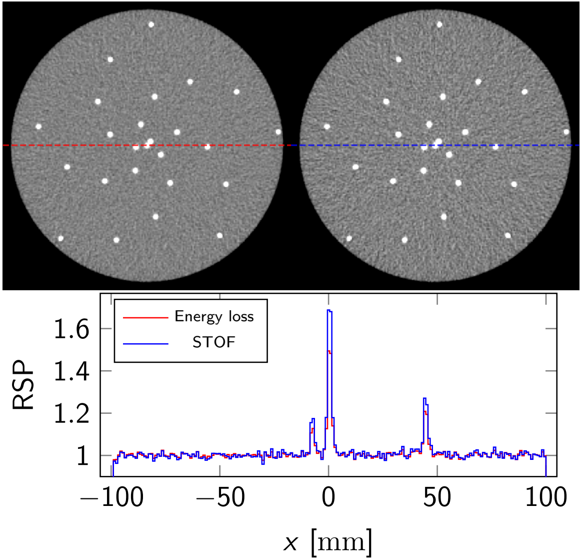


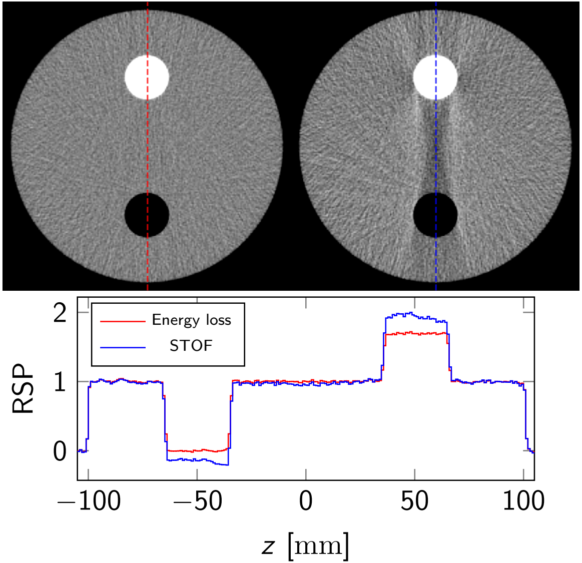


# Tomographic reconstructions (energy loss/sandwich time-of-flight)



# Profiles


Energy loss
STOF
RSP
1.6
1.4
1.2
1
−100
−50
0
50
100
$x$ [mm]


Energy loss
STOF
RSP
2
1
0
−100
−50
0
50
100
$z$ [mm]

# Advantages and limitations of this method

- Advantages:
  - Trivial to implement
  - Minimal computational cost
- Limitations:
  - Can only accurately reconstruct water
  - Assumes that the phantom is centered at the isocenter

# Advantages and limitations of this method

- Advantages:
    - Trivial to implement
    - Minimal computational cost
- Limitations:
    - Can only accurately reconstruct water
    - Assumes that the phantom is centered at the isocenter

→Need for a more advanced reconstruction technique!

# Outline

1. Introduction: sandwich time-of-flight pCT
2. Forward model and gradient descent reconstruction
3. Results
4. Conclusion

# Outline of the method

1. Start with an estimate of the RSP distribution
2. Calculate what would be the measured TOFs assuming this estimate (forward model)
3. Measure a distance between calculated TOFs and measured TOFs (cost function)
4. Use this metric to update the estimate accordingly (gradient descent)
5. Start over from 1

# Forward model

## Continuous version

- TOF of proton $p$:

$$\tau_p = \int_{\Gamma_p} \frac{\mathrm{d}\boldsymbol{x}}{v_p(\boldsymbol{x})}$$

- Velocity of the proton $p$ at point $\boldsymbol{x}$:

$$v_p(\boldsymbol{x}) = c^2 \frac{E_p(\boldsymbol{x})}{E_p(\boldsymbol{x}) + mc^2} \sqrt{1 + 2\frac{mc^2}{E_p(\boldsymbol{x})}}$$

- Energy of the proton $p$ at point $\boldsymbol{x} = \Gamma_p(s)$:

$$E_p(\Gamma_p(s)) = E_{\text{in}} - \int_{-\infty}^{s} \rho(\Gamma_p(s'))\sigma_{\mathrm{H_2O}}(E_p(\Gamma_p(s')))\mathrm{d}s'$$

# Forward model

## Continuous version

- TOF of proton $p$:

$$\tau_p = \int_{\Gamma_p} \frac{\mathrm{d}\boldsymbol{x}}{v_p(\boldsymbol{x})}$$

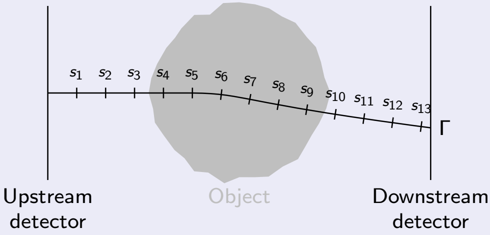
- Velocity of the proton $p$ at point $\boldsymbol{x}$:

$$v_p(\boldsymbol{x}) = c^2 \frac{E_p(\boldsymbol{x})}{E_p(\boldsymbol{x}) + mc^2} \sqrt{1 + 2\frac{mc^2}{E_p(\boldsymbol{x})}}$$

- Energy of the proton $p$ at point $\boldsymbol{x} = \Gamma_p(s)$:

$$E_p(\Gamma_p(s)) = E_{\text{in}} - \int_{-\infty}^{s} \rho(\Gamma_p(s'))\sigma_{\mathrm{H_2O}}(E_p(\Gamma_p(s')))\mathrm{d}s'$$

## Discrete version



# Forward model

## Continuous version

- TOF of proton $p$:

$$\tau_p = \int_{\Gamma_p} \frac{\mathrm{d}\boldsymbol{x}}{v_p(\boldsymbol{x})}$$

- Velocity of the proton $p$ at point $\boldsymbol{x}$:

$$v_p(\boldsymbol{x}) = c^2 \frac{E_p(\boldsymbol{x})}{E_p(\boldsymbol{x}) + mc^2} \sqrt{1 + 2\frac{mc^2}{E_p(\boldsymbol{x})}}$$

- Energy of the proton $p$ at point $\boldsymbol{x} = \Gamma_p(s)$:

$$\begin{aligned} E_p(\Gamma_p(s)) =& E_{\text{in}} - \\ & \int_{-\infty}^{s} \rho(\Gamma_p(s'))\sigma_{\mathrm{H_2O}}(E_p(\Gamma_p(s')))\mathrm{d}s' \end{aligned}$$

## Discrete version

- TOF of proton $p$:

$$\tau_p \approx \sum_{k=1}^{K} \frac{d_p^{s_k}}{v_p^{s_k}} \tag{4}$$

- Velocity of the proton $p$ at point $\boldsymbol{x}$:

$$v_p^{s_k} = c^2 \frac{E_p^{s_k}}{E_p^{s_k} + mc^2} \sqrt{1 + 2\frac{mc^2}{E_p^{s_k}}} \tag{5}$$

- Energy of the proton $p$ at point $\boldsymbol{x} = \Gamma_p(s_k)$:

$$E_p^{s_k} = E_p^{s_{k-1}} - d_p^{s_k}\tilde{\rho}_\rho(\Gamma_p(s_{k-1}))\sigma_{\mathrm{H_2O}}(E_p^{s_{k-1}}) \tag{6}$$

# Optimization scheme

- Let
  - $\hat{\boldsymbol{\tau}}\left(\boldsymbol{\rho}\right)$ the TOFs produced by the forward model with RSP map $\boldsymbol{\rho}$
  - $\boldsymbol{\tau}$ the measurements

# Optimization scheme

- Let
  - $\hat{\boldsymbol{\tau}}\left(\boldsymbol{\rho}\right)$ the TOFs produced by the forward model with RSP map $\boldsymbol{\rho}$
  - $\boldsymbol{\tau}$ the measurements
- The cost function is defined as

$$\mathcal{L}\left(\boldsymbol{\rho}\right) = \left\|\hat{\boldsymbol{\tau}}\left(\boldsymbol{\rho}\right) - \boldsymbol{\tau}\right\|_2^2 \tag{7}$$

# Minimization of the cost function

$$\mathcal{L}\left(\boldsymbol{\rho}\right) = \left\|\hat{\boldsymbol{\tau}}\left(\boldsymbol{\rho}\right) - \boldsymbol{\tau}\right\|_2^2$$

- The cost function can be minimized using gradient descent
- In its simplest form, the update step is given by $\rho_i' = \rho_i - \alpha \dfrac{\partial \mathcal{L}}{\partial \rho_i}$
- The derivative is computed using *automatic differenciation*

# Automatic differentiation and the chain rule

Automatic differentiation works by repeatedly applying the chain rule $\left(\frac{\partial x}{\partial z} = \frac{\partial x}{\partial y}\frac{\partial y}{\partial z}\right)$:

- $\frac{\partial \mathcal{L}}{\partial \rho_i} = \left(\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{\tau}}}\right)^\top \frac{\partial \hat{\boldsymbol{\tau}}}{\partial \rho_i}$

# Automatic differentiation and the chain rule

Automatic differentiation works by repeatedly applying the chain rule $\left(\frac{\partial x}{\partial z} = \frac{\partial x}{\partial y}\frac{\partial y}{\partial z}\right)$:

- $\frac{\partial \mathcal{L}}{\partial \rho_i} = \left(\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{\tau}}}\right)^\top \frac{\partial \hat{\boldsymbol{\tau}}}{\partial \rho_i}$
- $\frac{\partial \hat{\tau}_p}{\partial \rho_i} = -\sum_{k=1}^{K} \frac{d_p^{s_k}}{(v_p^{s_k})^2} \frac{\partial v_p^{s_k}}{\partial \rho_i}$

# Automatic differentiation and the chain rule

Automatic differentiation works by repeatedly applying the chain rule $\left(\frac{\partial x}{\partial z} = \frac{\partial x}{\partial y}\frac{\partial y}{\partial z}\right)$:

- $\dfrac{\partial \mathcal{L}}{\partial \rho_i} = \left(\dfrac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{\tau}}}\right)^\top \dfrac{\partial \hat{\boldsymbol{\tau}}}{\partial \rho_i}$
- $\dfrac{\partial \hat{\tau}_p}{\partial \rho_i} = -\sum_{k=1}^{K} \frac{d_p^{s_k}}{(v_p^{s_k})^2} \dfrac{\partial v_p^{s_k}}{\partial \rho_i}$
- $\dfrac{\partial v_p^{s_k}}{\partial \rho_i} = c^2 \left( \frac{mc^2}{(E_p^{s_k}+mc^2)^2} \sqrt{1 + 2\frac{mc^2}{E_p^{s_k}}} + \frac{mc^2}{\sqrt{1+2\frac{mc^2}{E_p^{s_k}}}} \frac{E_p^{s_k}}{E_p^{s_k}+mc^2} \right) \dfrac{\partial E_p^{s_k}}{\partial \rho_i}$

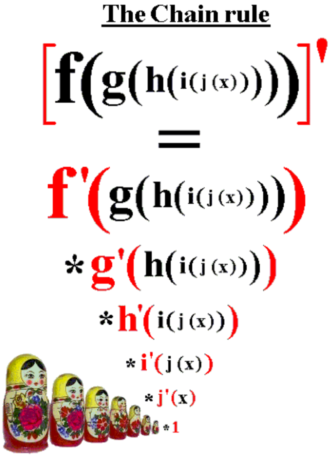
# Automatic differentiation and the chain rule

Automatic differentiation works by repeatedly applying the chain rule $\left(\frac{\partial x}{\partial z} = \frac{\partial x}{\partial y}\frac{\partial y}{\partial z}\right)$:

- $$\frac{\partial \mathcal{L}}{\partial \rho_i} = \left(\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{\tau}}}\right)^\top \frac{\partial \hat{\boldsymbol{\tau}}}{\partial \rho_i}$$
- $$\frac{\partial \hat{\tau}_p}{\partial \rho_i} = -\sum_{k=1}^{K} \frac{d_p^{s_k}}{(v_p^{s_k})^2} \frac{\partial v_p^{s_k}}{\partial \rho_i}$$
- $$\frac{\partial v_p^{s_k}}{\partial \rho_i} = c^2 \left( \frac{mc^2}{(E_p^{s_k}+mc^2)^2} \sqrt{1 + 2\frac{mc^2}{E_p^{s_k}}} + \frac{mc^2}{\sqrt{1+2\frac{mc^2}{E_p^{s_k}}}} \frac{E_p^{s_k}}{E_p^{s_k}+mc^2} \right) \frac{\partial E_p^{s_k}}{\partial \rho_i}$$
- $$\frac{\partial E_p^{s_k}}{\partial \rho_i} =$$
  $$\frac{\partial E_p^{s_{k-1}}}{\partial \rho_i} - d_p^{s_k} \left( \frac{\partial \tilde{\rho}_{\boldsymbol{\rho}}(\Gamma_p(s_{k-1}))}{\partial \rho_i} \sigma_{\mathrm{H_2O}}(E_p^{s_{k-1}}) + \tilde{\rho}_{\boldsymbol{\rho}}(\Gamma_p(s_{k-1})) \frac{\partial \sigma_{\mathrm{H_2O}}(E_p^{s_{k-1}})}{\partial E_p^{s_{k-1}}} \frac{\partial E_p^{s_{k-1}}}{\partial \rho_i} \right)$$

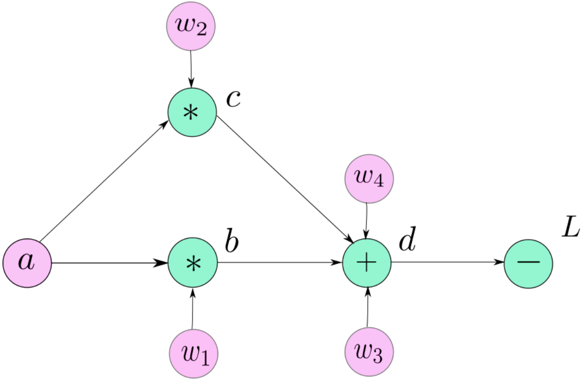
# PyTorch's automatic differentiation engine

- PyTorch is a free C++ deep learning library with Python bindings
- PyTorch provides an automatic differenciation engine (mainly for its backpropagation algorithm)



# Calculating the derivative with automatic differenciation

- PyTorch's `autograd` builds a graph representing the computation
- The gradients are computed using a `backward` pass

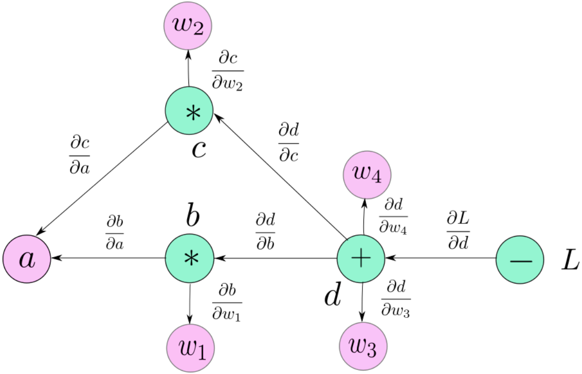


https://www.digitalocean.com/community/tutorials/pytorch-101-understanding-graphs-and-automatic-differentiation

```
import torch
a = 5.
w = torch.tensor([1., 2., 3., 4.],
 ↪  requires_grad=True)
b = a * w[0]
c = a * w[1]
d = b + c + w[2] + w[3]
L = -d
print(L)  # tensor(-22.,
 ↪  grad_fn=<NegBackward0>)
L.backward()
print(w.grad)  # tensor([-5., -5.,
 ↪  -1., -1.])
```

# Calculating the derivative with automatic differenciation

- PyTorch's autograd builds a graph representing the computation
- The gradients are computed using a backward pass


$w_2$
$\frac{\partial c}{\partial w_2}$
$*$
$c$
$\frac{\partial c}{\partial a}$
$\frac{\partial d}{\partial c}$
$w_4$
$b$
$\frac{\partial b}{\partial a}$
$\frac{\partial d}{\partial b}$
$\frac{\partial d}{\partial w_4}$
$\frac{\partial L}{\partial d}$
$a$
$*$
$+$
$-$
$L$
$\frac{\partial b}{\partial w_1}$
$d$
$\frac{\partial d}{\partial w_3}$
$w_1$
$w_3$

https://www.digitalocean.com/community/tutorials/pytorch-101-understanding-graphs-and-automatic-differentiation

```
import torch
a = 5.
w = torch.tensor([1., 2., 3., 4.],
 ↪  requires_grad=True)
b = a * w[0]
c = a * w[1]
d = b + c + w[2] + w[3]
L = -d
print(L)  # tensor(-22.,
 ↪  grad_fn=<NegBackward0>)
L.backward()
print(w.grad)  # tensor([-5., -5.,
 ↪  -1., -1.])
```

# Outline

1. Introduction: sandwich time-of-flight pCT

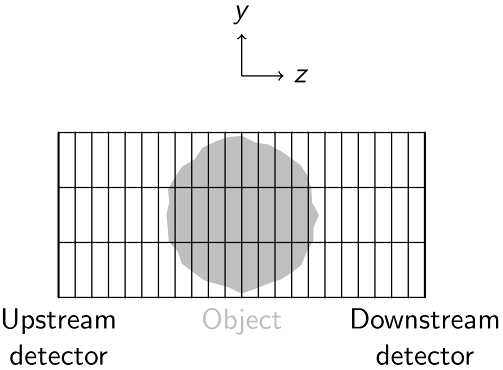
2. Forward model and gradient descent reconstruction

3. Results

4. Conclusion

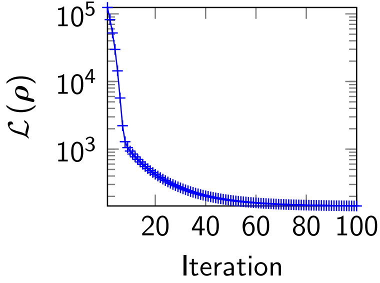
# Parameters

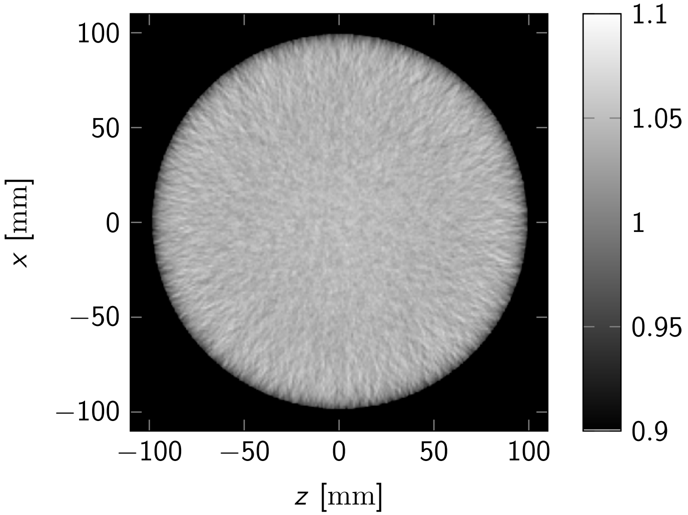
- Optimizer: Adagrad[4]
- Learning rate: 0.3
- Proton step size: 1 mm
- Grid size: 220 × 3 × 220
- Voxel size:
  $1.00\,\mathrm{mm} \times 133.33\,\mathrm{mm} \times 1.00\,\mathrm{mm}$
- Number of projections: 360
- Detected protons per cm$^2$: about 36.4



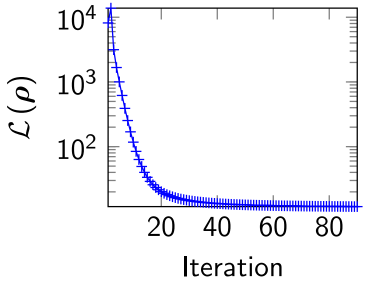
[4] John Duchi, Elad Hazan, and Yoram Singer. In: *Journal of machine learning research* 12.7 (2011)

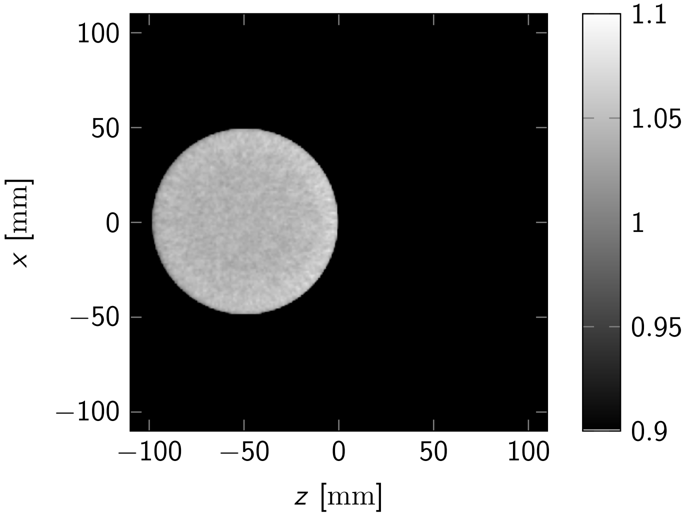
# Preliminary results: water phantom



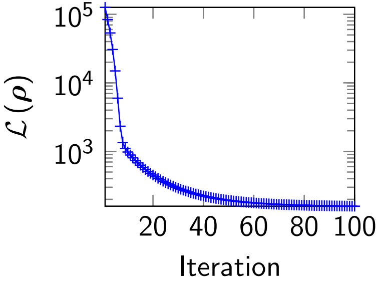


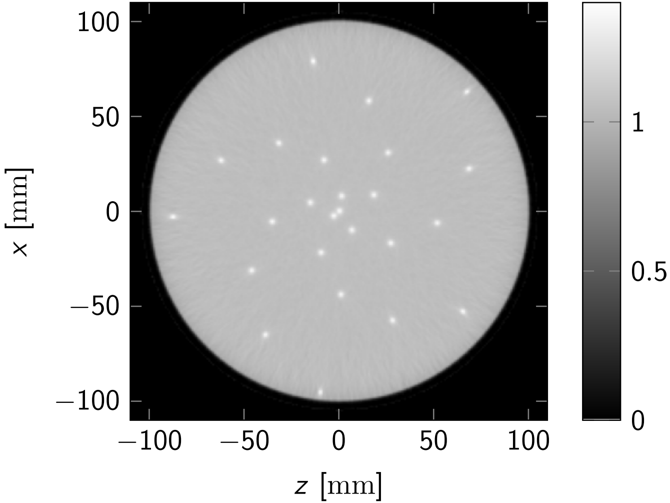
# Preliminary results: water phantom (off-centered)



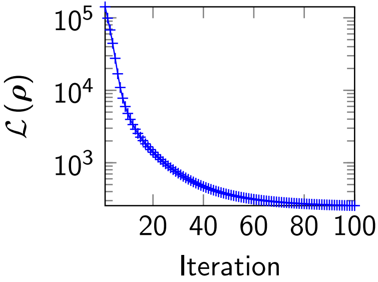


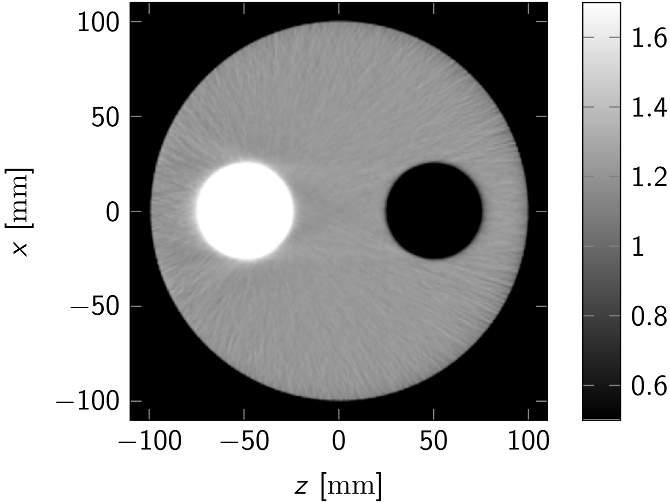
# Preliminary results: spiral phantom



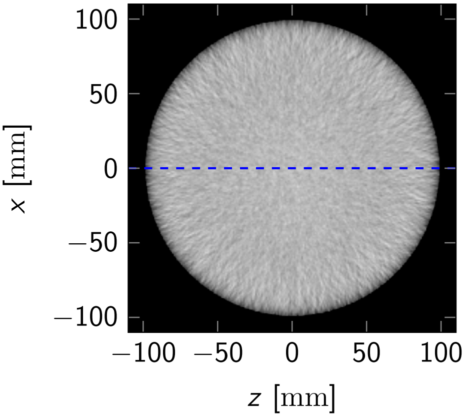


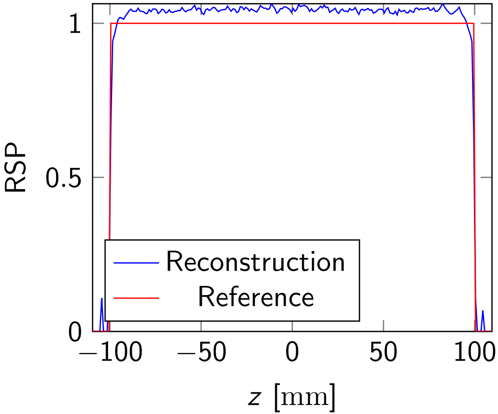
# Preliminary results: bone phantom



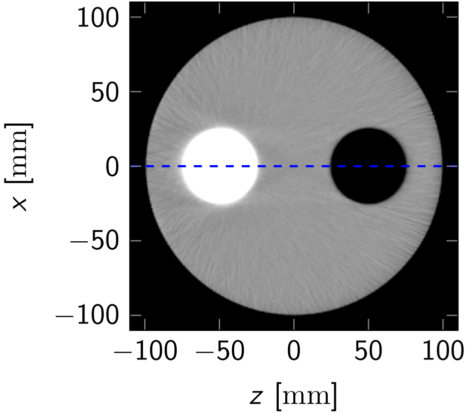


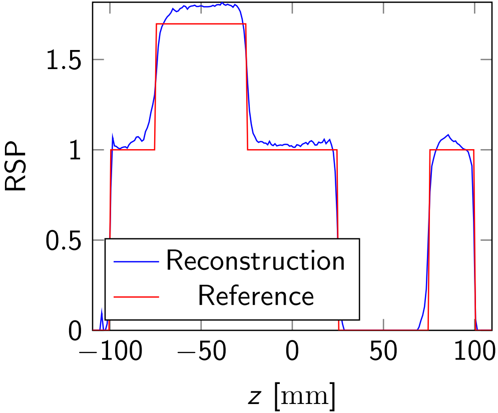
# Profile: water phantom





# Profile: bone phantom





# Advantages and limitations of this method

- Advantages:
  - Can theoretically reconstruct everything (assuming enough data)
  - Can incorporate a priori information about the object
    - Object support
    - Total variation
    - ...

# Advantages and limitations of this method

- Advantages:
  - Can theoretically reconstruct everything (assuming enough data)
  - Can incorporate a priori information about the object
    - Object support
    - Total variation
    - ...
- Limitations:
  - Computational cost are very high
  - Requires to manually tweak some hyperparameters
    - Learning rate
    - Step size along MLPs
    - Number of iterations
    - Optimizer
    - ...
  - Hard to debug!

# Outline

1. Introduction: sandwich time-of-flight pCT

2. Forward model and gradient descent reconstruction

3. Results

4. Conclusion

# Conclusion

- Iterative reconstruction of the RSP distribution seems possible, at a high computational cost
- The corresponding code is being contributed to PCT[5]
- Additional studies are necessary to properly assess the influence of each hyperparameter
- Real data would allow evaluating the method on more realistic assumptions

[5] https://github.com/acoussat/PCT/blob/pcttofopti/applications/pcttofopti/pcttofopti.py

# Conclusion

- Iterative reconstruction of the RSP distribution seems possible, at a high computational cost
- The corresponding code is being contributed to PCT[5]
- Additional studies are necessary to properly assess the influence of each hyperparameter
- Real data would allow evaluating the method on more realistic assumptions

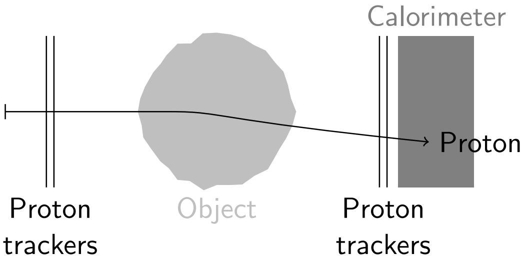
Grazie!

[5] https://github.com/acoussat/PCT/blob/pcttofopti/applications/pcttofopti/pcttofopti.py

# Backup slides

# Energy-loss proton computed tomography scanner

- Up- and downstream detectors track the protons' position and direction to estimate their MLP
- The calorimeter measures the residual energy to estimate the integral of the stopping power along the MLP
- The object rotates to acquire several projections



# Software environment

- Monte Carlo simulations were performed using GATE version 10[6]

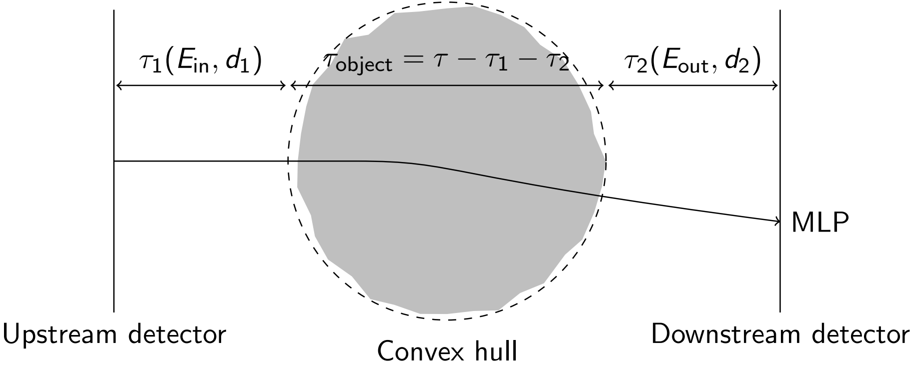
- Reconstructions were performed with PCT
  - Based on ITK and RTK[7]
  - Development on GitHub: `https://github.com/RTKConsortium/PCT`
  - More details in Simon Rit's talk!

[6] Nils Krah et al. In: *XXth International Conference on the use of Computers in Radiation therapy*. 2024

[7] Simon Rit et al. In: *Journal of Physics: Conference Series*. Vol. 489. 1. IOP Publishing. 2014

# 2D lookup table approach

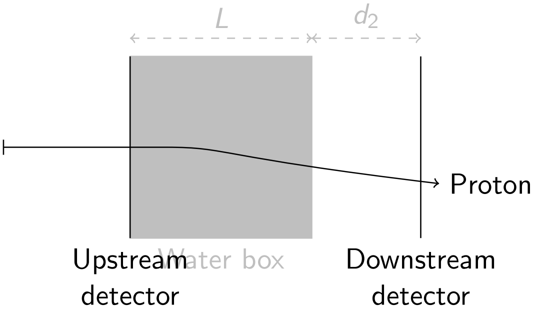
- Use the convex hull of the object to account for the air around the object
- Fit is constructed on the TOF between the beginning of the object and the downstream detectors, and the amount of air between the end of the object and the downstream detector



# The fit

The Monte Carlo is repeated for...

- 10 realisations of $d_2$ from 0 to 220 mm
- 10 realisations of $L$ from 0 to $220 - d_2$



$L$

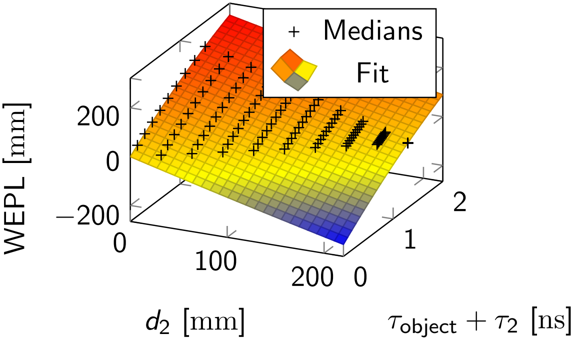
$d_2$

Proton

Upstream detector

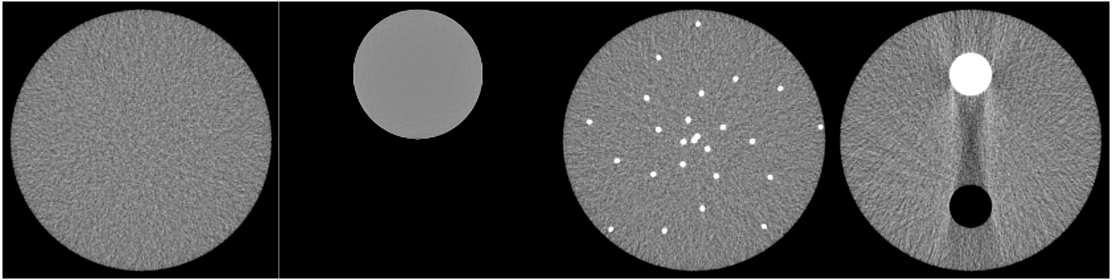
Water box

Downstream detector

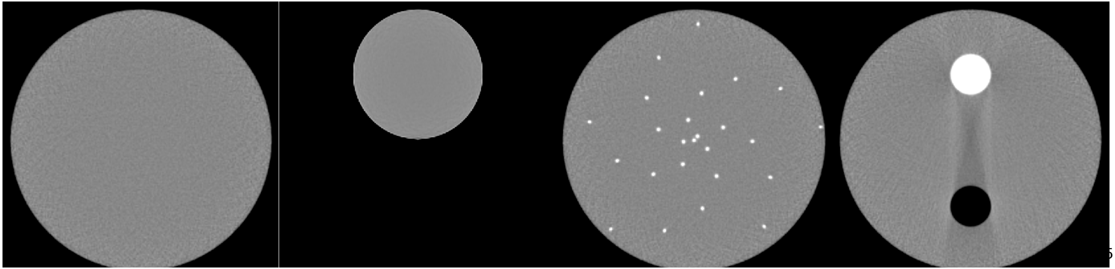


# Reconstructions

Previous method:

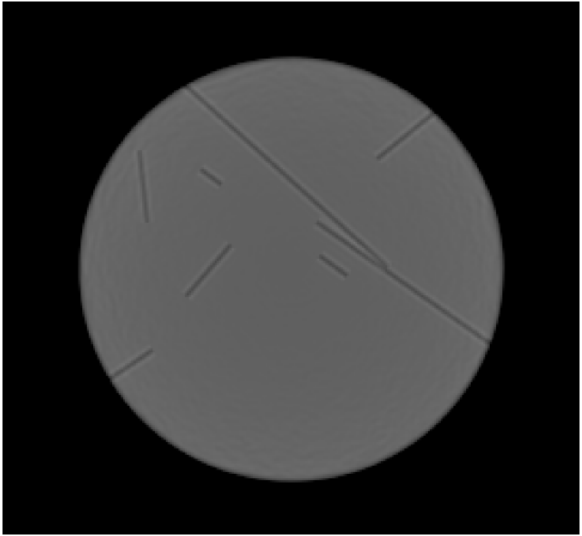


New method:



# One issue with this method?

- The method does not accound for the length of the MLP inside the object
- Therefore it cannot distinguish if the object is “empty” along the MLP
- In other words, it assumes that the convex hull is perfectly known
- Can this be improved?



# Calculating the derivative

## Analytical differenciation

By repeatedly applying the chain rule:

- $$\frac{\partial \mathcal{L}}{\partial \rho_i} = \left(\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{\tau}}}\right)^\top \frac{\partial \hat{\boldsymbol{\tau}}}{\partial \rho_i}$$
- $$\frac{\partial \hat{\tau}_p}{\partial \rho_i} = -\sum_{k=1}^{K} \frac{d_p^{s_k}}{(v_p^{s_k})^2} \frac{\partial v_p^{s_k}}{\partial \rho_i}$$
- $$\frac{\partial v_p^{s_k}}{\partial \rho_i} = c^2 \left( \frac{mc^2}{(E_p^{s_k} + mc^2)^2} \sqrt{1 + 2\frac{mc^2}{E_p^{s_k}}} + \frac{mc^2}{\sqrt{1 + 2\frac{mc^2}{E_p^{s_k}}}} \frac{E_p^{s_k}}{E_p^{s_k} + mc^2} \right) \frac{\partial E_p^{s_k}}{\partial \rho_i}$$
- $$\frac{\partial E_p^{s_k}}{\partial \rho_i} = \frac{\partial E_p^{s_{k-1}}}{\partial \rho_i} - d_p^{s_k} \left( \frac{\partial \tilde{\rho}_{\boldsymbol{\rho}}(\Gamma_p(s_{k-1}))}{\partial \rho_i} \sigma_{\mathrm{H_2O}}(E_p^{s_{k-1}}) + \tilde{\rho}_{\boldsymbol{\rho}}(\Gamma_p(s_{k-1})) \frac{\partial \sigma_{\mathrm{H_2O}}(E_p^{s_{k-1}})}{\partial E_p^{s_{k-1}}} \frac{\partial E_p^{s_{k-1}}}{\partial \rho_i} \right)$$

# Calculating the derivative

## Analytical differenciation

```
$E_p \leftarrow E_{\text{in}}$    ▷ Current energy
$\dfrac{\partial E_p}{\partial \rho_i} \leftarrow 0$    ▷ Current derivative of the energy
$\dfrac{\partial \hat{\tau}_p}{\partial \rho_i} \leftarrow 0$    ▷ Current TOF derivative
for $k \leftarrow 2, \ldots, K$ do
    $\dfrac{\partial E_p}{\partial \rho_i} \leftarrow \dfrac{\partial E_p}{\partial \rho_i} - d_p^{s_k} \left( \phi_i(\Gamma_p(s_k)) \sigma_{\mathrm{H_2O}}(E_p) + \tilde{\rho}_{\boldsymbol{\rho}}(\Gamma_p(s_{k-1})) \dfrac{\partial \sigma_{\mathrm{H_2O}}(E_p)}{\partial E_p} \dfrac{\partial E_p}{\partial \rho_i} \right)$
    $\dfrac{\partial v_p}{\partial \rho_i} \leftarrow c^2 \left( \dfrac{mc^2}{(E_p + mc^2)^2} \sqrt{1 + 2\dfrac{mc^2}{E_p}} + \dfrac{mc^2}{\sqrt{1 + 2\frac{mc^2}{E_p}}} \dfrac{E_p}{E_p + mc^2} \right) \dfrac{\partial E_p}{\partial \rho_i}$
    $\dfrac{\partial \hat{\tau}_p}{\partial \rho_i} \leftarrow \dfrac{\partial \hat{\tau}_p}{\partial \rho_i} - \dfrac{d_p^{s_k}}{v_p^2} \dfrac{\partial v_p}{\partial \rho_i}$    ▷ Update TOF derivative
    $E_p \leftarrow E_p - d_p^{s_k} \tilde{\rho}_{\boldsymbol{\rho}}(\Gamma_p(s_{k-1})) \sigma_{\mathrm{H_2O}}(E_p^{s_k - 1})$    ▷ Update current energy
end for
return $\dfrac{\partial \hat{\tau}_p}{\partial \rho_i}$
```

# Calculating the derivative

Finite differences

$$\frac{\partial \mathcal{L}}{\partial \rho_i} = \lim_{h \to 0} \frac{\mathcal{L}\left(\begin{bmatrix} \rho_0 \\ \rho_1 \\ \vdots \\ \rho_i + h \\ \vdots \\ \rho_J \end{bmatrix}\right) - \mathcal{L}\left(\begin{bmatrix} \rho_0 \\ \rho_1 \\ \vdots \\ \rho_i \\ \vdots \\ \rho_J \end{bmatrix}\right)}{h} \tag{8}$$

Super slow but useful nonetheless to check other implementations!

# autograd custom functions

- Possibility to replace some parts of the gradient graph with custom analytical derivatives
- Some advantages:
  - Reduced memory footprint
  - Reduced computation time (?)
  - Validation of the analytical derivatives using `torch.autograd.gradcheck`
- Main drawback: additional complexity

```
class Velocity(torch.autograd.Function):

    @staticmethod
    def forward(ctx, e):
        ctx.save_for_backward(e)
        return c * (e / (e + m0)) * torch.sqrt(1 +
        ↪  2 * (m0 / e))

    @staticmethod
    def backward(ctx, grad_output):
        e, = ctx.saved_tensors
        dv_de = -c*e*torch.sqrt(1 + 2*m0/e)/(e +
        ↪  m0)**2 + c*torch.sqrt(1 + 2*m0/e)/(e +
        ↪  m0) - c*m0/(e*torch.sqrt(1 +
        ↪  2*m0/e)*(e + m0))
        return grad_output * dv_de
```

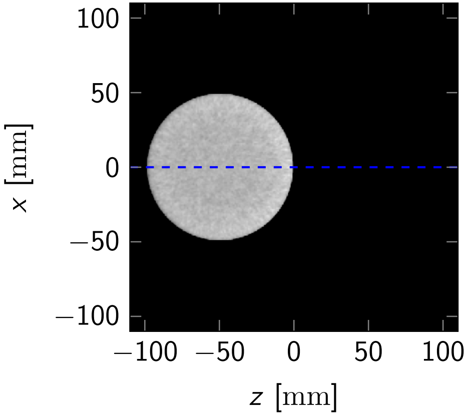
# Computation costs

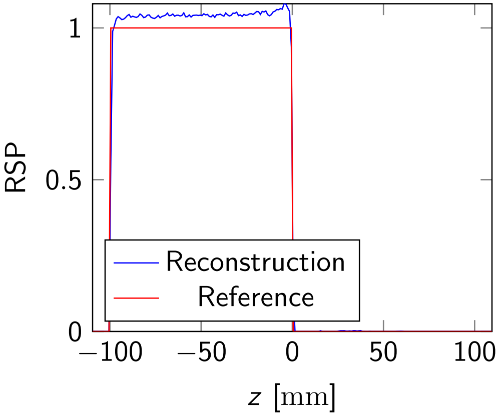
| | **Speed** | **Memory usage** |
|---:|---|---|
| `autograd` | 16 h | 20 GB |
| `autograd` + GPU | 1 h | 20 GB |
| `autograd` + functions | 16 h | 10 GB |
| `autograd` + GPU + functions | 1 h | 10 GB |

Why not have the whole forward model as a single `autograd` operator?

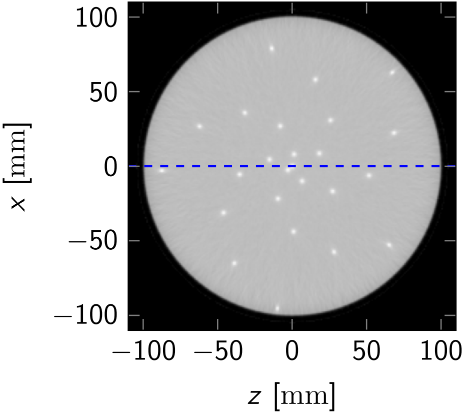
How to compute $\dfrac{\partial \rho_i}{\partial \hat{\tau}_p}$?

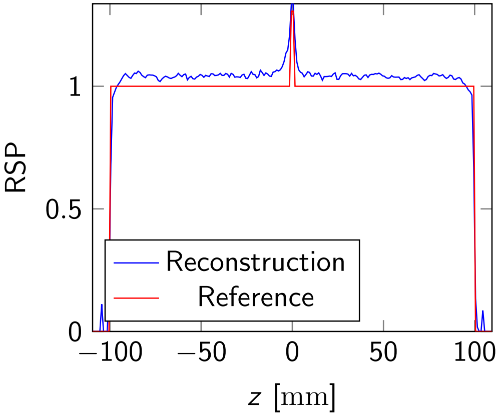
# Profile: water phantom (off-centered)





# Profile: spiral phantom




RSP
z [mm]
Reconstruction
Reference