

# Student Project Python CUDA C++ Bindings

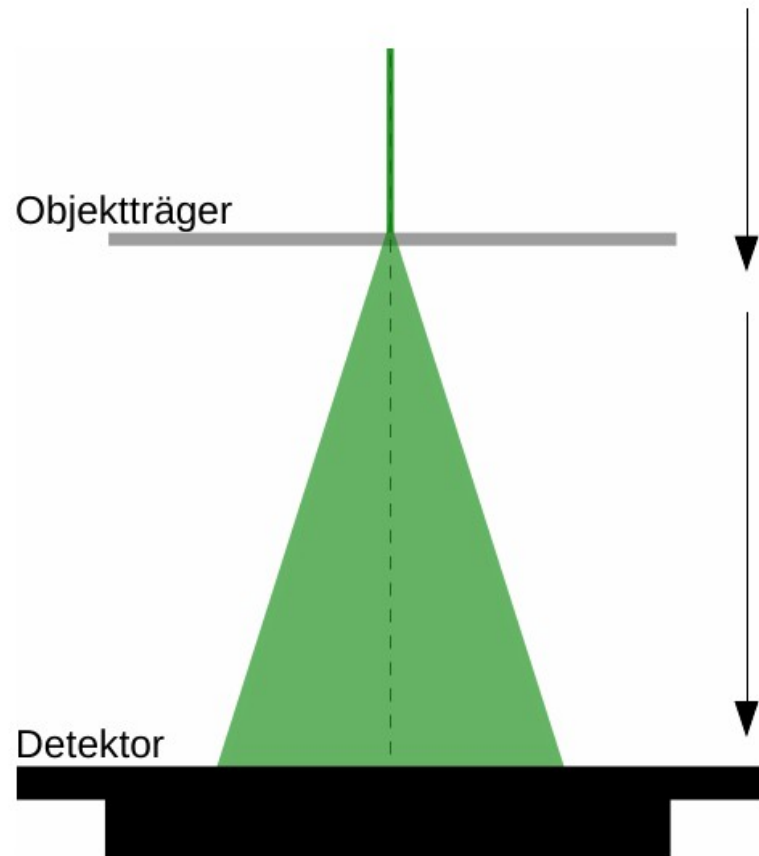
S.Ehrig

December 2020



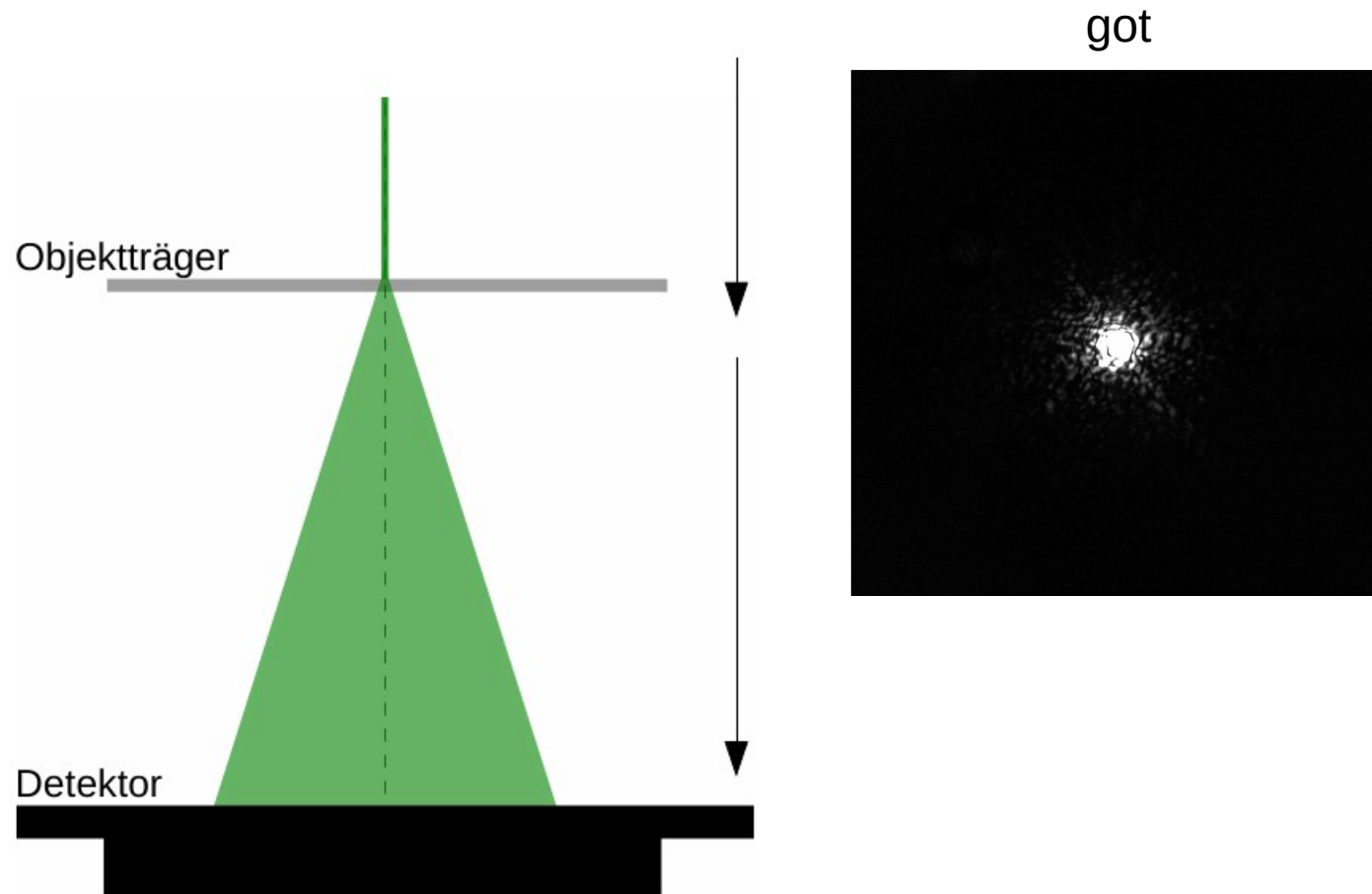
# Motivation

Ptychography – lensless microscopy



# Motivation

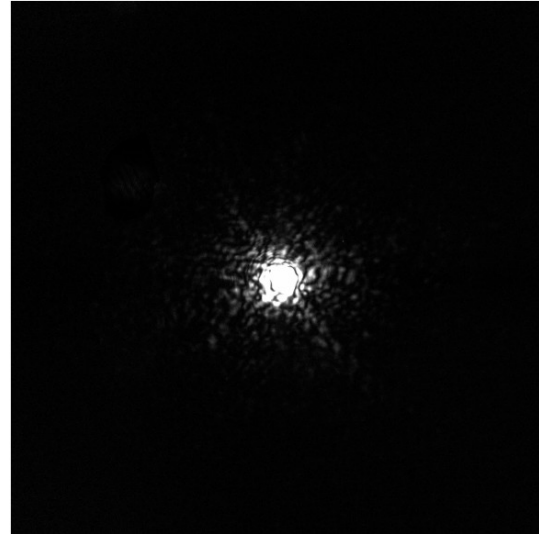
Ptychography – lensless microscopy



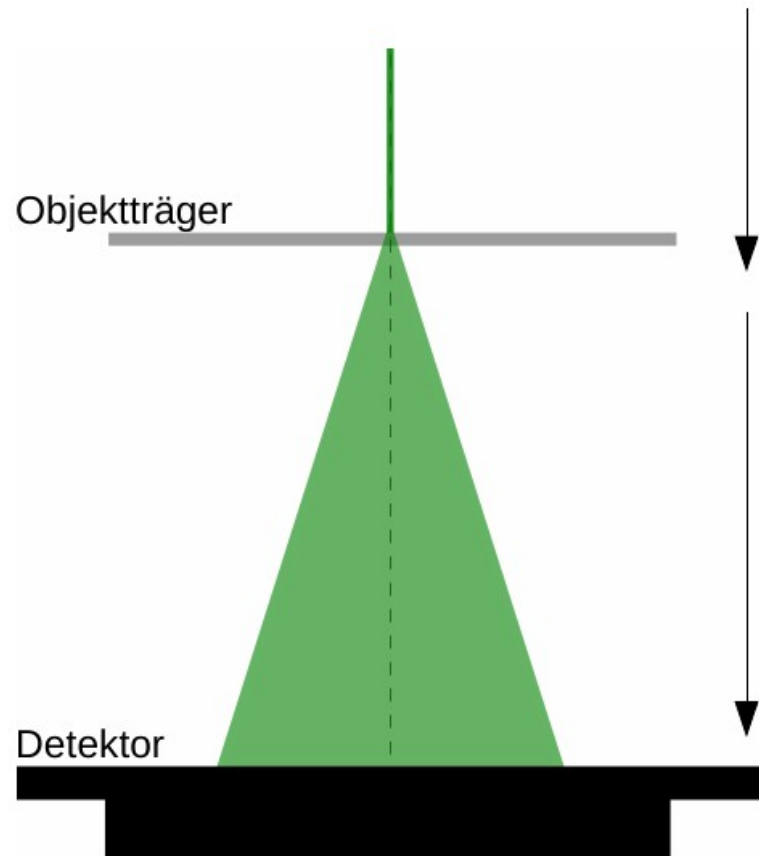
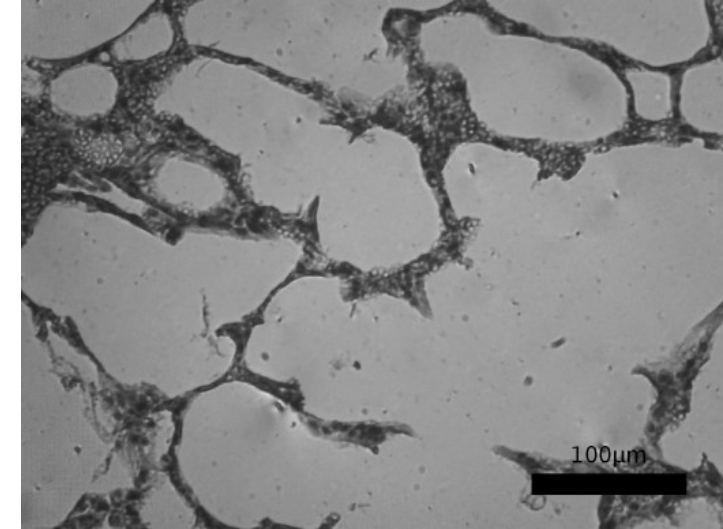
# Motivation

Ptychography – lensless microscopy

got

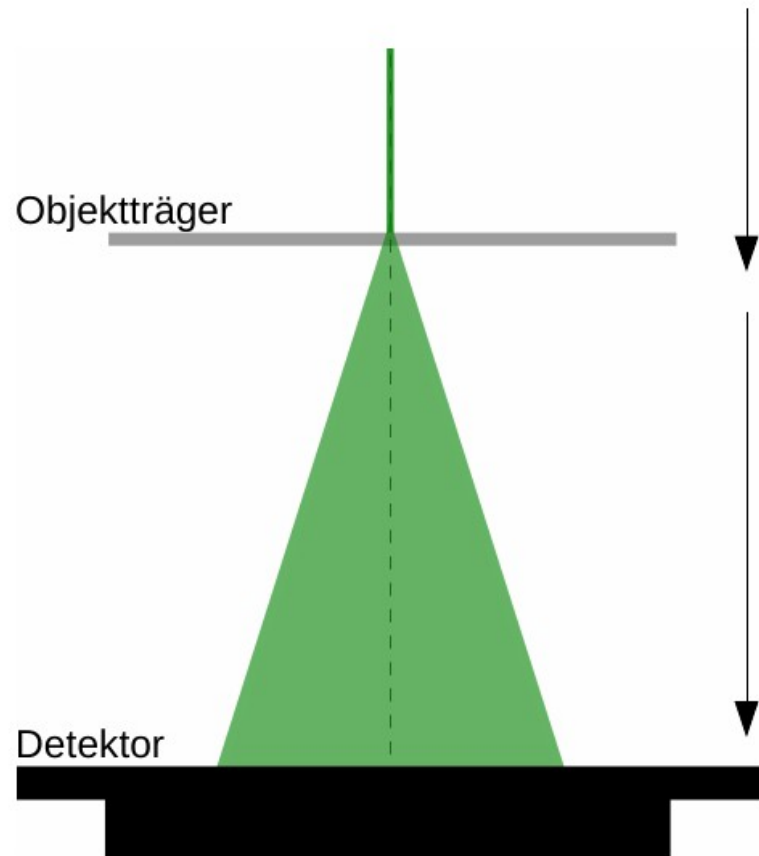


wanted

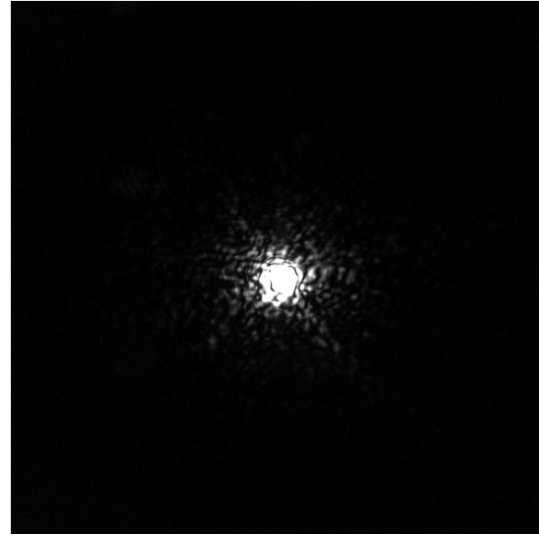


# Motivation

Ptychography – lensless microscopy

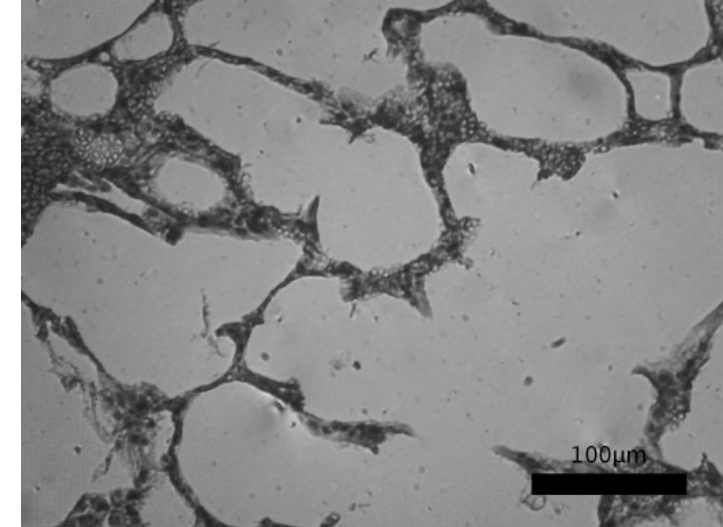


got



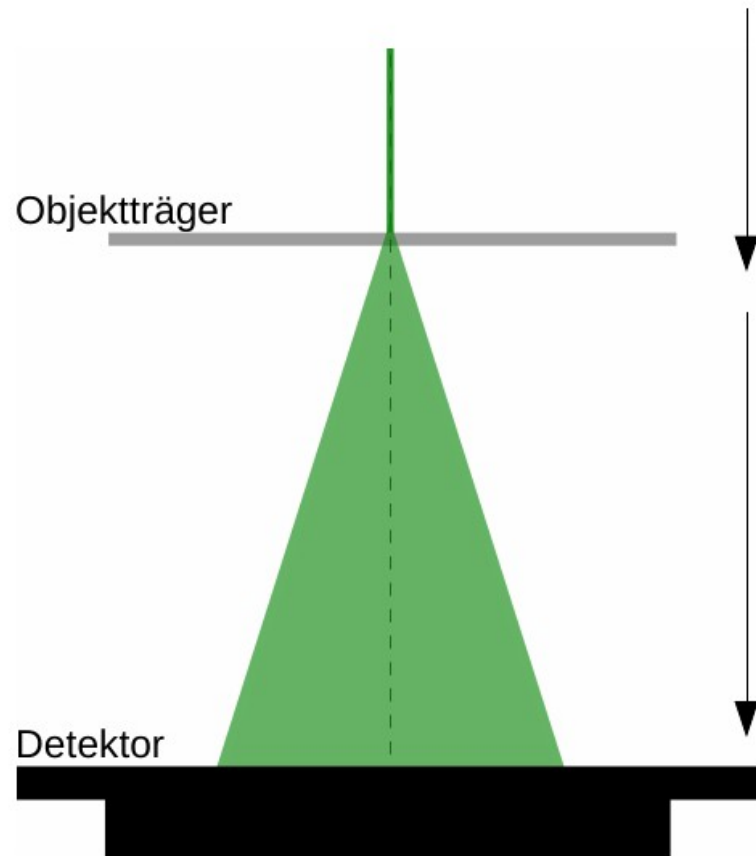
?

wanted

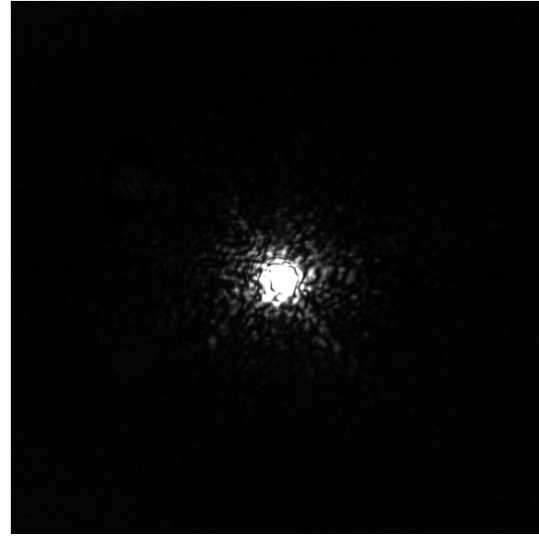


# Motivation

Ptychography – lensless microscopy

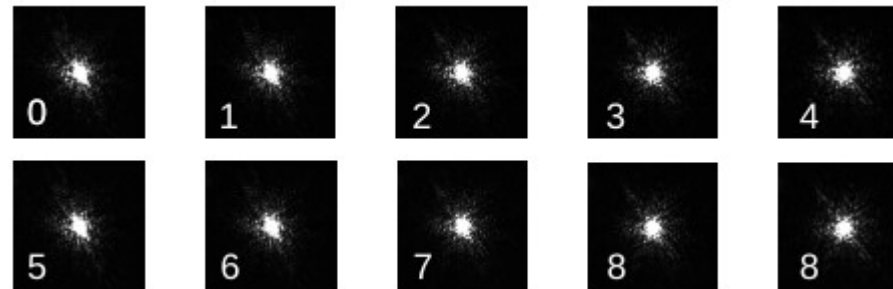
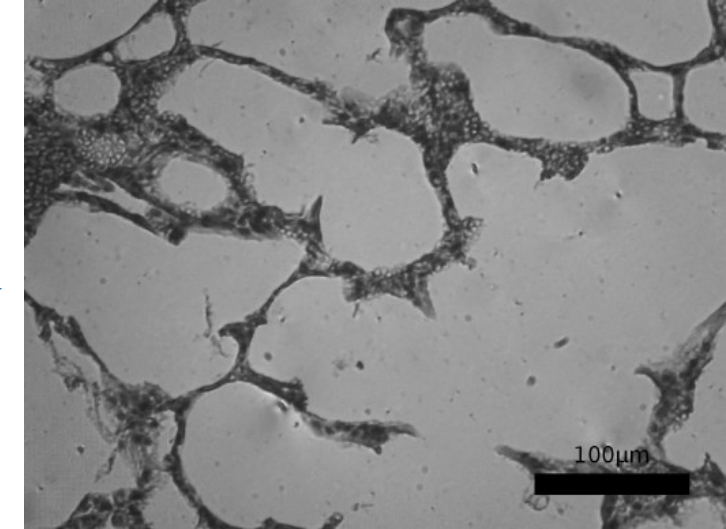


got



?

wanted



# Motivation

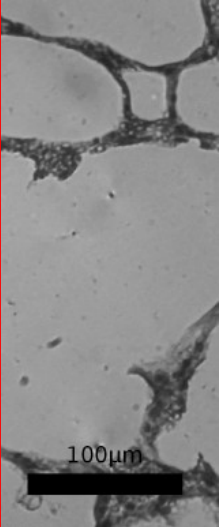
Ptychography – lensless microscopy

## Developing library for Ptychography

- Python Interface for easy reconstruction
- Kernels implemented in C++ to run fast on GPU und CPU (alpaka – accelerator abstraction library)
- Python bindings with pybind11

Objekttr

Detekto



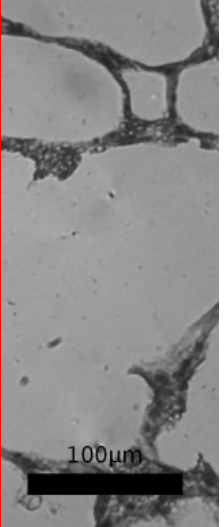
# Motivation

Ptychography – lensless microscopy

## Developing library for Ptychography

- Python Interface for easy reconstruction
- Kernels implemented in C++ to run fast on GPU und CPU (alpaka – accelerator abstraction library)
- Python bindings with pybind11

**Missing experience for effective usage of pybind11 with GPUs**



Objektttr

Detekto





# Tasks

- Implement the algorithm in CUDA C++ and add Python bindings

# Tasks

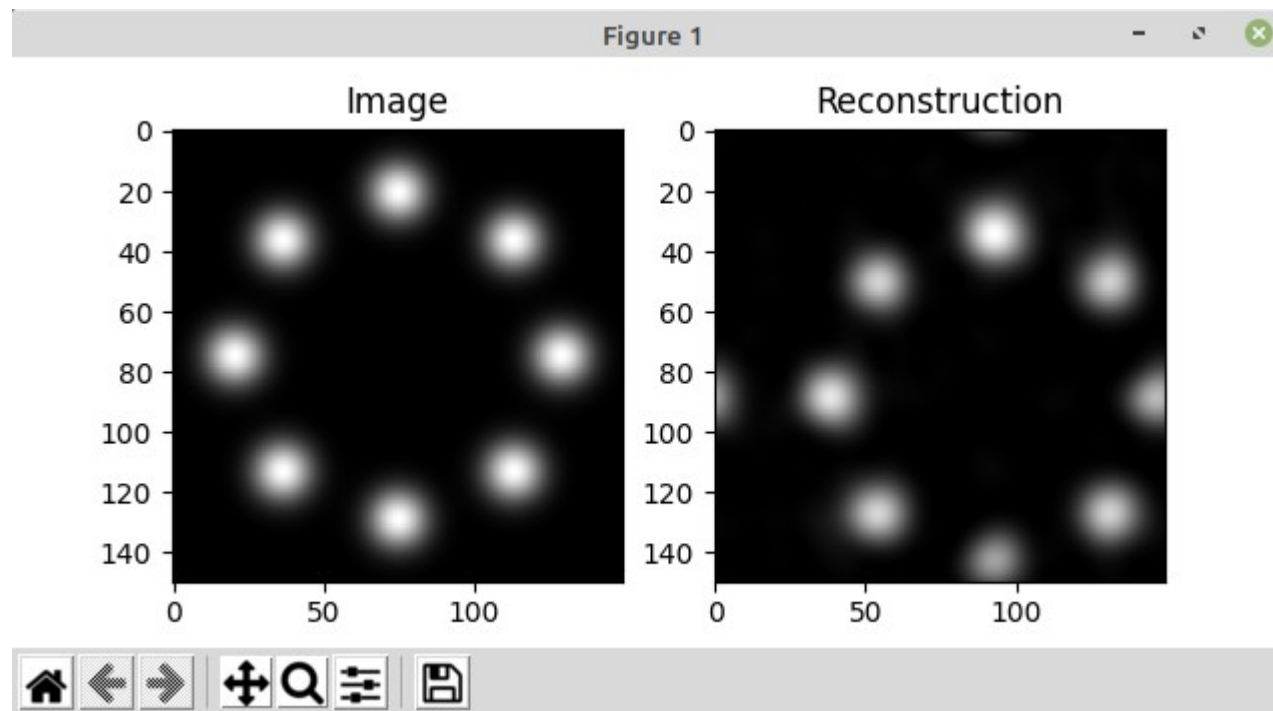
- Implement the algorithm in CUDA C++ and add Python bindings
- Develop Python bindings with pybind11 in different fashions
  - a single Python function call for the main loop
  - m Python function calls for each iteration
  - $m \cdot n$  Python function calls for each kernel

# Tasks

- Implement the algorithm in CUDA C++ and add Python bindings
- Develop Python bindings with pybind11 in different fashions
  - a single Python function call for the main loop
  - m Python function calls for each iteration
  - m\*n Python function calls for each kernel
- analyze performance overhead of the Python bindings
- Analyze GPU memory behavior (ownership and copies)
- Optimize code

# Algorithm

- Gerchberg–Saxton algorithm
- Reconstruction of greyscale images only from measurements of the magnitudes.
- Python implementation exists: <https://github.com/tuelwer/phase-retrieval>



# Algorithm

```
def Gerchberg-Saxton(Magnitude, Initial_Phase, numIters = 1000):  
    "  
    Magnitude: experimentally acquired diffraction image  
    Initial_Phase: first guess of phase  
  
    returns:  
        A: real-domain reconstruction  
    "  
  
    A := Magnitude × exp(i × Initial_Phase)  
    while i in range(numIters):  
        B := IFT(A)  
        B[B<0] = 0  
  
        C := FT(B)  
        A := Magnitude × exp(i × Phase(C))  
  
    return A
```