Glimpse MRI

September 2, 2025

Project Summary

Glimpse MRI is a Windows High-Performance Computing (HPC) application for visualizing MRI scans and exporting them to common image formats. Built in modern C++/Qt and accelerated with NVIDIA CUDA, it supports imaging R&D teams working on MRI algorithms, hardware, and methodologies. Written in C++ with Qt 6, Glimpse MRI offloads compute-intensive MRI stages—coil noise pre-whitening, CG-SENSE reconstruction, and post-filtering—to NVIDIA GPUs via CUDA (cuFFT, cuBLAS, custom kernels). It reads DICOM and HDF5 (fastMRI/ISMRMRD), offers slice-wise inspection/export, and includes hooks for parameter sweeps and PSNR/SSIM-style QA to quantify improvements. Glimpse MRI is a Windows Model–View–Controller (MVC) application, built with Qt, for viewing and converting MRI images. It supports both DICOM and raw MRI data in fastMRI and ISMRMRD HDF5 formats, with CUDA-accelerated reconstruction on the GPU. **Note:** For research/education only—not for clinical use.

Overview

- Windows HPC application for visualizing MRI scans and exporting to common image formats.
- Built in C++/Qt; accelerated with NVIDIA CUDA for MRI R&D workflows.
- GPU-offloaded stages: coil noise pre-whitening, CG-SENSE reconstruction, post-filtering (cuFFT, cuBLAS, custom kernels).
- Reads DICOM and HDF5 (fastMRI / ISMRMRD); slice-wise inspection/export; hooks for parameter sweeps and QA.
- **QA will be performed with PSNR and SSIM**.
- Research/education tool; not for clinical use.

Features

- **File support**
  - DICOM images
  - HDF5 (.h5) datasets in fastMRI and ISMRMRD formats
- **Reconstruction and processing**
  - Coil noise pre-whitening
  - CG-SENSE reconstruction
  - 2D spatial filtering (extensible pipeline)

  - GPU acceleration via CUDA (cuFFT, cuBLAS, custom kernels)
- **Experimentation and QA**
  - Parameter sweeps for algorithms
  - PSNR/SSIM comparisons against references
  - Structured debug logs for reproducibility
- **User interface**
  - Simple Qt GUI
  - Interactive zoom and slice navigation (Arrow keys + Ctrl)
  - Context menu to export PNG or BMP

Architecture

1. **Back-end (Model)** — `mri_engine.dll`
   - Reads .h5 (fastMRI / ISMRMRD)
   - Pre-processing: coil noise pre-whitening
   - Reconstruction: CG-SENSE
   - Post-processing: 2D spatial filters
   - Heavy compute on GPU (CUDA)
2. **Front-end (View)**
   - Qt GUI for visualization
   - Displays reconstructed images
   - Tools: zoom, slice scrolling, export
3. **Controller**
   - Orchestrates Model ↔ View
   - HDF5 (.h5): calls `mri_engine` → GPU processing → returns OpenCV `cv::Mat` → converted to `QImage` for display
   - DICOM: uses standard C++ DICOM libraries to parse and display
   - Coordinates image saving/export

Tech Stack

- Language: C++
- Framework: Qt 6 (GUI)
- GPU: CUDA (cuFFT, cuBLAS, custom kernels)
- Imaging: OpenCV
- Formats: DICOM, fastMRI, ISMRMRD

Agile / Kanban Plan

Columns

- Done

- In Progress
- Next Up (First Release)
- Backlog (Second Release and later)
- QA (PSNR/SSIM)

Done

Iteration 1 (completed)

- Model: `mri_engine` with FFT-only reconstruction (baseline).
- View: simple Qt GUI; context menu for saving PNG and DICOM.
- Controller: orchestrates I/O (HDF5 fastMRI only; hardcoded image path).
- Tested: Qt app displays one slice; writes PNG and DICOM to disk.

In Progress

- Integrate DICOM reading alongside fastMRI.
- Add ISMRMRD reading path and Controller plumbing.
- Slice visualization controls (scrubber/keyboard) for multi-slice series.
- GPU acceleration harness for compute-intensive tasks (wire up cuFFT/cuBLAS and kernels).
- QA harness: compute PSNR and SSIM vs reference images (e.g., RSS full-k).

Next Up (First Release scope)

- File support: DICOM, fastMRI, ISMRMRD.
- Slice visualization of MRI images.
- GPU acceleration of compute-intensive tasks.
- Saving as DICOM, PNG, and BMP.
- CG-SENSE reconstruction.
- 2D spatial median filtering.
- Pre-whitening using algorithm in `mri/prewhiten.hpp`
  - Uses `mri/io.hpp` and `mri/common.hpp`
  - Cholesky-based whitener from k-space corner covariance
  - CPU: `apply_whitener_cpu`; CUDA: `apply_whitener_cuda`
- QA: PSNR and SSIM for each pipeline configuration; include debug logs and parameter dumps.

Backlog (Second Release and later)

- CPU multithreaded implementations for parity with GPU paths.
- CG-SENSE-TV (total variation regularization).
- Improved pre-whitening algorithm (e.g., shrinkage-regularized covariance, eigenvalue clipping).
- Better spatial filtering (e.g., bilateral/NLM or guided filter) to replace median baseline.
- Extended QA dashboards: per-slice PSNR/SSIM tables and CSV export.
- Robust file handling: non-hardcoded paths, drag-and-drop, batch processing.

Working Agreements

- WIP limit: 2 items per active column to maintain flow.
- Every feature includes basic debug prints (start/end, sizes, parameters, timings).
- QA gate: image-quality features must report PSNR and SSIM before moving to **Done**.