Objective:

- Restructure the `darkline_pointer` codebase to improve memory management, readability, and efficiency.
- Replace outdated memory handling with standard library containers (e.g., `std::vector`) and simplify the line removal methods.

Activities:

- **Header File (darkline_pointer.h):**
    - Replaced `DarkLinePtrArray` with `std::vector<DarkLine>` for storing detected dark lines.
    - Retained `RemovalMethod` enum values (`DIRECT_STITCH` and `NEIGHBOR_VALUES`).
    - Preserved `DarkLine` struct with members: `x`, `y`, `startX`, `startY`, `endX`, `endY`, `width`, `isVertical`, and `inObject`.
- **Implementation File (darkline_pointer.cpp):**
    - **Memory Management & Resizing:**
        - Replaced `new` and `delete[]` with `std::vector` for automatic memory management.
        - Used `std::copy` for efficient data transfer during the dark line removal process.
    - **Parallelization:**
        - Refactored to use `std::vector<std::thread>` instead of `std::thread` and `std::mutex`, simplifying parallel processing for line detection.
    - **Removal Methods:**
        - Retained the two removal methods (`DIRECT_STITCH` and `NEIGHBOR_VALUES`) with a cleaner implementation.
    - **Selective Removal:**
        - Integrated selective removal logic directly into `removeDarkLinesSequential`, making it more streamlined and efficient.
    - **Helper Functions:**

        - Retained helper functions (`isInObject`, `calculateSearchRadius`, `findReplacementValue`, `findStitchValue`) with minor improvements in readability.
    - **Consolidated Removal Functions:**
        - Reduced three separate removal functions (`removeDarkLines`, `removeDarkLinesSelective`, `removeAllDarkLines`) into two functions (`removeDarkLines` and `removeDarkLinesSequential`), centralizing the selective removal logic.
    - **Naming Conventions:**
        - Updated naming by replacing `DarkLinePtrArray` and `DarkLinePtr` with `std::vector<DarkLine>` and the `DarkLine` struct directly.

Achievements:

- Improved memory management using `std::vector`, reducing manual memory allocation.
- Simplified and modernized codebase, making it easier to maintain and understand.
- Enhanced parallelization implementation for better CPU utilization.

Problems & Solutions:

- **Problem:** Complexity in memory management with `new` and `delete[]` operations.
    - **Solution:** Replaced with `std::vector` for automatic handling, removing the need for manual memory allocation and deallocation.
- **Problem:** Code complexity due to separate functions for selective and general dark line removal.
    - **Solution:** Consolidated functions to streamline the code, integrating selective removal into `removeDarkLinesSequential`.
- **Problem:** Old parallelization approach required multiple `std::thread` and `std::mutex` implementations, making it challenging to manage.
    - **Solution:** Used `std::vector<std::thread>` for cleaner and more efficient parallelization.
- **Problem:** Mixed data handling between double 2D pointers and vectors, with `ImageData` struct relying on double pointers.
    - **Solution:** Confirmed use of `std::vector` for `DarkLine` storage, pending decision on standardizing `ImageData` storage method.