GPU Image Filtering & Thermal Visualization

CUDA + OpenCV Capstone Project

Author: Abhinith

Course: GPU Specialization Capstone

Project Objectives

- Demonstrate CUDA kernel programming for per-pixel operations.
- Implement GPU reductions (min/max) and normalization.
- Apply a thermal colormap LUT on GPU for visualization.
- Integrate CUDA code with OpenCV for I/O.
- Deliver reproducible build & run instructions.

Processing Pipeline

1. Read input image (OpenCV)
2. Convert BGR → Grayscale (CUDA kernel)
3. Find min/max via parallel reduction (CUDA)
4. Normalize pixels to [0,1] (CUDA)
5. Apply thermal colormap LUT (CUDA)
6. Save output (OpenCV)

CUDA Kernels & Implementation

Grayscale kernel uses coalesced access to read BGR and compute: $Y = 0.299 R + 0.587 G + 0.114 B$ Reduction uses shared memory per-block then global combine. Normalization is a simple per-pixel affine transform. Colormap LUT maps normalized intensity to RGB (thermal).

Memory & Performance Considerations

- Use global memory for image buffers (device).
- Use shared memory to speed up reductions per block.
- Minimize host-device transfers; keep work on GPU.
- Occupancy tuned with block sizes; coalescing accesses.

Parallel Reduction & Normalization

Reduction finds min and max in $O(n)$ work across threads. After reduction, normalization per pixel: $(v - min)/(max - min)$. This avoids clipping and improves contrast for thermal mapping.

Thermal Colormap

Normalized value (0..1) maps to a thermal colormap LUT on GPU. Colors progress: blue → cyan → yellow → red. Result: visual emphasis of hot regions and gradients.

Build & Run (Windows, VS + CUDA)

```
Open 'x64 Native Tools Command Prompt for VS 2022' cd /d
C:\Users\abhin\OneDrive\Desktop\CUDA-Image-Processing-Capstone-Project cmake -S . -B build
-DOpenCV_DIR="C:/Users/abhin/Downloads/opencv/build" cmake --build build --config Release cd build\Release
GPU_Image_Filter.exe ..\..\input\input.jpg ..\..\output\output.png
```

Results & Metrics

- Runtime: Milliseconds for 1024x1024 images (GPU dependent)
- Memory: Single pass, minimal host-device transfers
- Quality: Thermal mapping improves visual contrast
- Presentation-ready images stored in /output folder

Future Work & Credits

- Add GPU Gaussian blur & Sobel edge detection (real-time).
- Real-time webcam thermal mode using OpenCV streaming.
- FFT-based denoising as an advanced filter option.
- Multi-GPU batching for larger datasets.

Credits: NVIDIA CUDA, OpenCV, Project Author: Abhinith