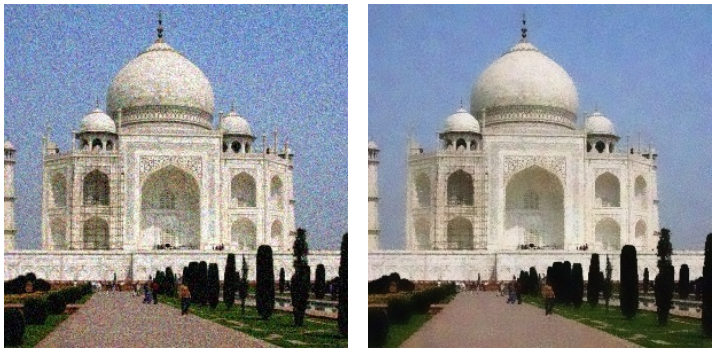
CMDA 4634 Fall 2024: Final Project Proposal

Xavier Akers
Computational Modeling and Data Analytics
Virginia Tech
xakers@vt.edu

April 27, 2025

1 Objective

I plan to implement and optimize an edge-persevering smoothing filter called bilateral filtering [4]. Bilateral filtering's main effort is reducing noise in images. In practice, this smoothing technique can be used for enhancing digital images, cleaning medical scans, rendering 3D graphics, and much more [3]. However, bilateral filtering is a computationally expensive technique, posing challenges when applied to high-resolution videos or real-time video streams. My goal is to improve the performance of the typical CPU-based implementation by leveraging the power of GPU accelerated computing.



(a) Original noisy image. (b) Image after bilateral filtering.

Figure 1: Example of bilateral filtering from GeeksforGeeks.org [1].

I will first implement a typical bilateral filtering implementation in C as a baseline for the algorithm's performance. I will use Python's efficient I/O to transform images into large matrices and pass them to a C shared object containing the algorithm. I will then extend this implementation with CUDA to accelerate the processing. Using CUDA provides the algorithm with the ability to assign each pixel to a separate thread. A major challenge arises, for each thread working on a pixel, that thread must gather information on the neighboring pixels. This requires optimizing memory management to take advantage of the GPU's architecture.

The next phase will focus on optimizing the GPU implementation for video processing. Videos consist of multiple frames, and each frame will be treated as a separate image. The high memory requirements for high-resolution and high-FPS videos will pose an issue for naive CUDA implementations. I will use CUDA streams and asynchronous Memcpys to develop an efficient system for streaming video frames to the device and back to the host [2].

Additionally, I will attempt to integrate a live webcam feed into the project, allowing the filter to be applied

to a real-time video input. This will show practical scenarios such as live broadcasting, where performance is crucial.

My goal is to produce a CUDA optimized implementation of bilateral filtering that can handle video inputs in real-time. The success of this project will be evaluated based on the performance gains achieved through GPU acceleration compared to the typical C implementation.

2 Motivations

For this project, I am motivated by the opportunity to apply my skills and experiences with large point sets to image and video processing. Processing images and videos are often computationally expensive, especially for high-resolution videos or real-time applications presenting the potential for accelerated GPU computing. Edge-preserving smoothing techniques have a large impacts in various fields such as medical, scientific imaging, computer vision and many more, all of which I hope to contribute to in the future.

3 Resources

I plan to use Google Collab, Pascal, and ARC.

4 Software

1. Python Packages
    i. Numpy
    ii. Matplotlib
    iii. Ctypes
    iv. PIL (for image I/O)
    v. OpenCV (for video I/O. Not currently installed on Pascal or ARC)
        - `pip install opencv-python`
2. C & CUDA Packages
    i. Standard C packages

5 Validation

Milestones

1. Basic Implementation of Bilateral Filtering on Images
    a. Deliverable: Implemented algorithm in C and CUDA for images (C implementation provides a benchmark).
2. Optimization and Memory Management
    a. Deliverable: Optimized CUDA code to improve computation and memory usage on images.
3. Tune for Video processing
    a. Deliverable: Adjusted code to process videos and provide more optimizations such as CUDA streams and asynchronous memory operations.
4. CUDA Stream Integration

a. Deliverable: Integrated CUDA streaming to processes frames concurrently.

5. Report, Repository, and Performance results

   a. Deliverable: Full report with references to code and performance results.

Stretch Goals	CUDA and Python code optimized to take in live webcam feed and broadcast the filtered video at a rate close to 1:1.
A	Bilateral filtering is implemented and fully optimized on both images and videos. Git repository is organized and code is readable and well documented. There is evidence of significant speedup. Presentation is descriptive and informative. Report demonstrates a high level understanding of GPU concepts such as kernel optimization, memory management and new topics not covered in class.
B	Bilateral filtering is implemented and somewhat optimized for images and videos. Code is accessible and somewhat readable and documented. There is only some evidence of speedup or evidence is not well presented. Presentation is clear but lacks some advanced details and descriptions. Report is missing key details of the GPU accelerated computing.
C	Bilateral filtering is not fully implemented, that is, it does not apply to both images and videos. Git repository is not organized or code is not readable or well documented. There is minimal evidence of speedup or evidence is not presented. Presentation only provides surface level explanation of the project. Report is cluttered and difficult to comprehend.

Table 1: Rubric

References

[1] GeeksforGeeks. *Python Bilateral Filtering - GeeksforGeeks*. [Online; accessed 08-October-2024. 2024. URL: https://www.geeksforgeeks.org/python-bilateral-filtering/.

[2] NVIDIA. *GPU Pro Tip: CUDA 7 Streams Simplify Concurrency*. https://developer.nvidia.com/blog/gpu-pro-tip-cuda-7-streams-simplify-concurrency/. Accessed: 2024-10-08. 2024.

[3] Sylvain Parais et al. "Bilateral Filtering: Theory and Application". In: *Foundations and TrendsR in Computer Graphics and Vision* 4.1 (2008), pp. 1–73. DOI: 10.1561/0600000020.

[4] Wikipedia contributors. *Bilateral filter — Wikipedia, The Free Encyclopedia*. [Online; accessed 08-October-2024]. 2024. URL: https://en.wikipedia.org/wiki/Bilateral_filter.