High Performance Computing
Deliverable 1: GPU Porting Decisions Report

This report outlines the decisions made regarding which functions of the Kanade-Lucas-Tomasi Tracker will be ported on the gpu for acceleration .The Klt algorithm estimates the 2D translation and scale changes of an image template between original template coordinates and a given reference image convolution, gradient, and interpolation operations.
In deliverable1 we profiled the CPU implementation using gprof and gprof2dot . The purpose of this report is to analyze and interpret the profiling data and justify which computationally extensive components should be ported to the gpu.

Profiling Summary

Profiling was done on example3 which represents the core klt pipeline. We did multiple profile runs using gprof and averaged the results to determine the most time consuming changes.

Function Name	% Time (Avg)	Self Time (Avg, s)	Total Time (Avg, s)	GPU Porting Type
_convolveImageHoriz	35.93%	0.07	0.00	Full
_convolveImageVert	25.00%	0.05	0.00	Full
_KLTTrackFeatures	79.33%	0.00	0.15	Partial
_convolveSeparate	60.93%	0.00	0.11	Full
_interpolate	19.47%	0.04	0.00	Full
_KLTComputeGradients	40.60%	0.00	0.08	Full
_KLTComputeSmoothedImage	20.30%	0.00	0.04	Full
_KLTSelectGoodFeatures	19.00%	0.02	0.02	Partial

Functions Selected for GPU Porting

1. **Convolution Functions(convolveSeperate , convolveImageHoriz ,convolveImageVert)**
   These dominate execution time, performing independent pixel-wise convolutions.
   Reason: Perfect for data-parallel GPU threads.
   GPU advantage: CUDA kernels can process horizontal and vertical convolutions concurrently.
2. **Optimization:** Use shared memory and texture caches to reduce latency.
   Ab dekho

3. **Interpolation (_interpolate)**
   Performs sub-pixel intensity estimation across millions of points.
   Reason: Each computation is independent.
   GPU advantage: Each pixel mapped to a GPU thread.
   Optimization: Texture memory enables faster caching and bilinear filtering.

4. **Gradient Computation (_KLTComputeGradients)**
   Computes image gradients (intensity change along x/y axes).
   Reason: Pixel-wise independent operations.
   GPU advantage: Parallel gradient computation for all pixels.
   Optimization: Shared memory reduces redundant global reads.

5. **Image Smoothing (_KLTComputeSmoothedImage)**
   Applies a low-pass filter to reduce noise.
   Reason: Each pixel filtered independently.
   GPU advantage: Threads process pixels in parallel.
   Optimization: Use separable filters and shared memory tiles.

6. **Feature Tracking and Selection (_KLTSelectGoodFeatures,KLTTrackFeatures)**
   Finds strong corner features via eigenvalue scoring.
   Reason: Scoring parallelizable; selection best on CPU.
   GPU advantage: GPU handles scoring; CPU finalizes selection.
   Optimization: Hybrid GPU-CPU processing balances speed and control flow.

Expected Impact

By porting the selected convolution and interpolation routines to GPU, significant speedup is expected, primarily due to:

- Reduced per-frame processing time for feature tracking.
- Improved throughput in real-time applications.
- Enhanced scalability for high-resolution video sequences.

Repository Link

All profiling data, source code, and GPU porting progress are maintained in the private GitHub repository: https://github.com/FatimaRana50/CS4110-klt-gpu

Conclusion

Profiling results clearly indicate that convolution-based image processing and interpolation routines consume the majority of execution time. These functions are **highly data-parallel** and **memory-access predictable**, making them excellent candidates for GPU acceleration. The decision to port these specific routines will lead to the most efficient utilization of GPU resources while maintaining manageable implementation complexity.