**HIGH PERFORMACE COMPUTING with GPUs**

**GROUP MEMBERS:**

**NABEEHA MAHMOOD 23i-0588**

**MAHAM FATIMA 23i-0685**

**Submitted to:**

**Dr. Imran Ashraf**

GITHUB REPO: <https://github.com/NabeehaMahmood/KLT-Feature-Tracker-GPU-Acceleration>

**COMPLEX COMPUTING PROBLEM**

**Deliverable #2**

**FALL 2025**

**Naïve GPU KLT Tracker — V2 Overview**

The GPU version (V2) of the KLT Feature Tracker was successfully implemented using CUDA. The same two 320×240 grayscale images were used as input to ensure consistency with the CPU baseline (V1). Both versions detected 100 good features; however, the GPU version tracked all 100 features successfully compared to 92 in the CPU version, likely due to minor floating-point precision differences in gradient computations.

The GPU implementation achieved a processing time of 0.0339 seconds, compared to 0.065 seconds for the CPU version, resulting in a speedup of approximately 2×. This confirms the correctness and efficiency of the GPU-based implementation.

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **V1 (CPU)** | **V2 (GPU)** | **Speedup** |
| **Image Size** | 320×240 | 320×240 | - |
| **Features Detected** | 100 | 100 | - |
| **Features Tracked** | 92 | 100 | - |
| **Lost Features** | 8 | 0 | - |
| **Processing Time (s)** | 0.065 | 0.0339 | 1.9× |
| **Features per Second** | 1538.46 | 2948.70 | 1.9× |

- **Goal:** Create a minimal, correct CUDA port of the V1 (CPU) KLT core.

- **Scope:** Cornerness (min-eigenvalue) and per-feature Lucas–Kanade tracking on GPU.

- **Non-goals** (deferred to V3): pyramids, shared memory, launch/occupancy tuning, texture memory, streams.

**1 Performance Methodology**

**1.1 Measurement**

  - End-to-end wall time measured using std::chrono::high\_resolution\_clock.

  - Timing includes gradient computation, feature detection, and tracking kernels.

  - Excludes initial H2D image transfers and final D2H feature copies for fair kernel comparison.

  - Run warm-ups; average over N runs.

**1.2 Baseline**

  - V1 (single-thread CPU) on same machine; focus on cornerness and LK step cost.

**1.3 Actual results (320×240 test case with img0.pgm and img1.pgm)**

  - V1 (CPU): 0.065 seconds, 92 features tracked

  - V2 (GPU): 0.034 seconds, 100 features tracked (0 lost)

  - Measured speedup: ~1.9× end-to-end

  - Features per second (V2): ~2,949 features/sec

  - The moderate speedup reflects the naive implementation with global memory access patterns and host-side feature selection overhead.

**2 Planned V3 Optimizations**

**- Launch configuration and occupancy**

  - Tune block sizes; use occupancy calculator; cap registers if needed.

**- Memory hierarchy**

  - Shared memory tiles for Ix/Iy and window sums

  - Read-only cache/texture binding for images

**- Communication**

  - Keep pyramids and gradients on device; device-side feature selection (thrust or custom)

  - Stream overlap for H2D/D2H and kernels

**- Algorithmic**

  - Build image pyramids on GPU; track coarse→fine

  - Optional smoothing using separable Gaussian kernels in CUDA

**3 Example Outputs**

- TXT files follow the KLT header format and list (x,y)=val per feature.

- PPM overlays draw red crosses at feature locations on each frame.

- Lost features marked as (-1,-1)=-1 in frame1 output.

**4 Kernel Analysis**

**4.1 computeGradients**

  - **Arithmetic intensity**: Low (2 FLOPs per 3 memory reads)

  - **Bottleneck**: Memory bandwidth

  - **Optimization potential**: Texture cache or shared memory prefetching

**4.2 computeFeatures**

  - **Arithmetic intensity**: Medium (window\_size² operations per pixel)

  - **Bottleneck**: Redundant global memory reads for overlapping windows

  - **Optimization potential**: Shared memory tiling with halo regions

**4.3 trackFeatures**

  - **Arithmetic intensity**: High (iterative solve with window accumulation)

  - **Bottleneck**: Thread divergence due to variable iteration counts

  - **Optimization potential**: Warp-level primitives, predicated execution

**Conclusion**

- V2 establishes a correct, working CUDA baseline for KLT's core steps.

- It provides measurable speedups on the parallel parts and a stable target for V3's optimizations.

- Actual testing confirms ~1.9× speedup over CPU baseline with 320×240 images and 100 features.

- All 100 features were successfully tracked with zero losses, demonstrating robust convergence in single-scale LK iteration.