**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 PROGRAMMING PROBLEM**

**Deliverable #1**

**FALL 2025**

**KLT Feature Tracker GPU Acceleration — V1 Analysis**  
The system tracks image features across frames, achieving 92/100 successful matches on 320×240 grayscale images in 0.065 s — a throughput of 1538 features/s.  
Identified bottlenecks are proposed for GPU acceleration in future versions (V2–V4).

**1. Baseline Implementation Analysis  
1.1 Algorithm Overview**

1. **Feature Detection** – Finds “good” corners via eigenvalue analysis (KLTSelectGoodFeatures()).
2. **Feature Tracking** – Tracks features using Lucas–Kanade optical flow (KLTTrackFeatures()).
3. **Pyramid Construction** – Builds Gaussian pyramids for multi-scale tracking (\_computePyramid()).

**1.2 Execution Output Summary**

|  |  |
| --- | --- |
| **Metric** | **Result** |
| Image Size | 320 × 240 |
| Features Selected | 100 |
| Features Tracked | 92 |
| Lost Features | 8 |
| Processing Time | 0.065 s |
| Throughput | 1538.46 features/sec |

These results establish the **CPU baseline** for performance comparison after GPU acceleration.

**1.3 Tracking Context Parameters**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Value** | **Description** |
| mindist | 10 | Minimum distance between features |
| window\_width, window\_height | 7 | Lucas–Kanade window size |
| smoothBeforeSelecting | TRUE | Apply smoothing before detection |
| min\_determinant | 0.01 | Threshold for feature corner strength |
| max\_iterations | 10 | Max iterations per feature |
| max\_residue | 10.0 | Tracking error threshold |
| pyramid\_levels | 2 | Multi-scale processing levels |

**2. Call Graph and Code Structure**:main()  
 ├ KLTCreateTrackingContext()  
 ├ KLTCreateFeatureList()  
 ├ pgmReadFile() ← Reads image files  
 ├ KLTSelectGoodFeatures() ← Feature detection  
 │ ├ \_computeGradients()  
 │ ├ \_computeHarrisResponse()  
 │ └ \_sortAndSelectFeatures()   
├ KLTWriteFeatureListToPPM() ← Visualization  
├ KLTTrackFeatures() ← Feature tracking  
 │ ├ \_computePyramid()  
 │ │ ├ \_convolveImageHoriz()  
 │ │ └ \_convolveImageVert()  
 │ ├ \_trackFeature()  
 │ └ \_refineFeatureLocation()  
├ KLTWriteFeatureListToPPM()  
 └ Cleanup (free memory)

**3. Profiling and Bottlenecks**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Description** | **Est. CPU Time %** | **GPU Priority** |
| KLTSelectGoodFeatures() | Pixel-wise eigenvalue computation | 35–40% | **High** |
| KLTTrackFeatures() | Iterative optical flow across features | 40–50% | **High** |
| \_computePyramid() | Multi-scale Gaussian smoothing | 10–15% | **Medium** |
| \_convolveImageHoriz/Vert() | Image convolution | 5–10% | **Medium** |

Profiling via gprof, manual timers, and later CUDA events will guide optimization.

**4. GPU Acceleration Strategy**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Parallelization Model** | **Justification** | **Expected Speedup** |
| **KLTSelectGoodFeatures()** | Thread-per-pixel | Independent pixel computations | 10–20× |
| **KLTTrackFeatures()** | Thread-per-feature | Independent optical flow tracking | 5–15× |
| **\_computePyramid()** | Thread-per-pixel + shared memory | Localized filtering | 5–10× |
| **\_convolveImageHoriz/Vert()** | Separable 1D convolution | Regular memory access | 3–8× |

**Memory Optimization**

* **Use pinned memory for images.**
* **Keep feature data on GPU between frames.**
* **Cache constants in constant memory.**
* **Employ shared and texture memory for filtering.**

**Challenges & Mitigations**

|  |  |  |
| --- | --- | --- |
| **Challenge** | **Description** | **Solution** |
| **Irregular Memory Access** | **Random feature positions** | **Texture caching** |
| **Load Imbalance** | **Varying per-feature work** | **Dynamic partitioning** |
| **Transfer Overhead** | **Host–device latency** | **Asynchronous streams** |

**6. Conclusion**V1 is a stable baseline. Prioritizing KLTSelectGoodFeatures() and KLTTrackFeatures() for GPU porting offers the largest potential gains. The staged approach preserves correctness while enabling progressive optimization and validation.