V2 Performance Report

Overview

This update introduces GPU acceleration for separable convolutions (horizontal and vertical) in the convolve_gpu.cu module. The CPU and GPU implementations now share the same Gaussian kernel generation, ensuring consistent results across both paths.

What Changed

• GPU Acceleration: Separable convolutions (horizontal and vertical) are now executed on the GPU via kernels in convolve_gpu.cu.

• Unified Gaussian Kernels: Both CPU and GPU paths use identical Gaussian weights generated by _KLTGetKernels.

• Consistent Public API: The public API functions _KLTComputeGradients and _KLTComputeSmoothedImage continue to be used, with backend selection at compile time.

• Build System Enhancements: The Makefile supports building both CPU (libklt.a) and GPU (libklt_gpu.a) variants, along with example3 and example3_gpu executables.

Test Setup

Example Used: example3 — default 320×240 image sequence from Version 1.

CPU Build: make && ./example3

GPU Build: make gpu NVCC=/usr/local/cuda/bin/nvcc && ./example3_gpu

Timing Tool: /usr/bin/time -f '%E real, %M KB maxrss'

Note: CUDA tools were unavailable on the current test system; only CPU results are recorded.

Results

Version	Command	Real Time	Max RSS	Comment
CPU	./example3	0.49 s	~5.4 MB	Works end-to-end
GPU	./example3_gpu	pending	pending	Run once CUDA tools are available

Current Bottlenecks

• Full Image Transfer Overhead: Each GPU call transfers the complete image to/from the device, which scales poorly with higher resolutions.

• Frequent Memory Allocation: Device memory is allocated and freed per call; persistent buffers would reduce overhead.

• Basic Kernel Configuration: Fixed 16×16 block layout; tuning and shared memory can improve performance.

• Synchronization Cost: Synchronizing after every vertical pass adds latency. CUDA streams/events can overlap operations.

Next Steps

• Benchmark GPU Performance: Run and record GPU timings on a CUDA-enabled system.

• Implement Buffer Reuse: Maintain device buffers across calls to reduce allocations.

• Optimize Kernel Launch Parameters: Tune block/grid dimensions and explore shared-memory tiling.

• Kernel Fusion: Combine horizontal and vertical passes to reduce bandwidth and synchronization costs.

• Profile Further Stages: Extend profiling to feature selection and tracking loops after validating convolution speedups.

Summary

This version establishes the foundation for GPU-accelerated convolution in the KLT pipeline. Although GPU performance data is pending, early architectural changes enable future scalability and performance improvements. The next development cycle will focus on optimization, buffer management, and profiling to maximize GPU acceleration benefits.