Real-time video processing

Visual Computing Assignment

Mads Pagh
Student ID: 202208375
Aarhus University
GitHub Repository

October 27, 2025

Contents

1 Methodology

1.1 System Architecture

The system consists of three main components:

- **Video Capture**: OpenCV VideoCapture for webcam input
- **Processing Pipeline**: CPU (OpenCV) and GPU (OpenGL/GLFW) implementations
- **Rendering**: OpenGL for display and GPU acceleration

1.2 Implemented Filters 4

1. **None** - The frames are passed through without any filtering
2. **Grayscale** - Color to grayscale conversion
3. **Gaussian Blur** - 5x5 kernel blur
4. **Edge Detection** - Sobel operator
5. **Pixelation** - 10x10 pixel blocks
6. **Comic Art** - Edge detection combined with color quantization

1.3 Geometric Transformations

Five transformation configurations were tested:

1. No Transform (baseline)
2. Translation only ($t_x = 0.3, t_y = 0.2$)
3. Scale only ($s = 1.5$)
4. Rotation only ($\theta = 25$ degrees)
5. Combined ($t_x = 0.2, t_y = -0.15, s = 1.3, \theta = 15$ degrees)

1.4 Test Resolutions

Three resolutions were benchmarked:

- VGA: 640×480 (307,200 pixels)
- HD: 1280×720 (921,600 pixels)
- Full HD: 1920×1080 (2,073,600 pixels)

1.5 Benchmarking Procedure

- Each test configuration ran for 50 frames (transforms) or 50 frames (resolution) due to hardware limitations
- Frame times were measured using high-resolution clock
- Average FPS and frame time statistics were calculated
- Tests were automated to ensure consistency, could have been done manually, but fear of human error and tendency to forget configurations made automation preferable

2 Results

2.1 Transform Benchmark Results

2.1.1 GPU vs CPU Performance

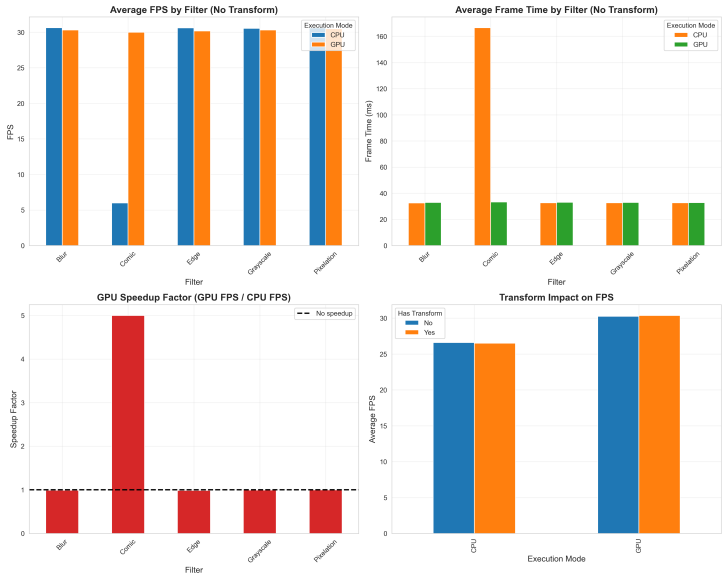


Figure 1: Performance comparison across filters and execution modes

It's clear that the GPU outperforms the CPU significantly only for the Comic Art filter, while for all other filters the performance is roughly equivalent, with the CPU even slightly outperforming the GPU in some cases. This suggests that for simple filters, the overhead of data transfer to the GPU negates any computational advantages.

2.1.2 Transform Impact

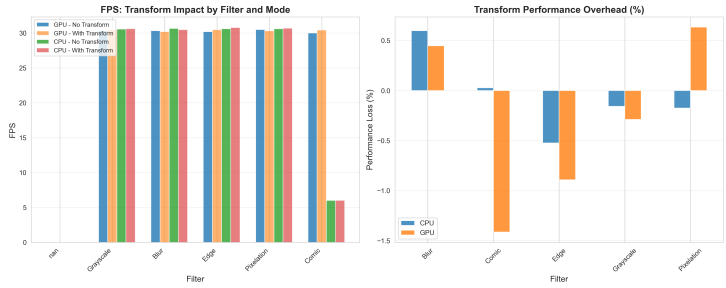


Figure 2: Impact of geometric transformations on performance

The results indicate that geometric transformations introduce minimal overhead on the GPU, with performance remaining nearly constant across all transform configurations. In contrast, CPU performance shows a slight decrease with transformations, likely due to the additional per-pixel computations required.

2.2 Resolution Benchmark Results

2.2.1 Resolution Scaling

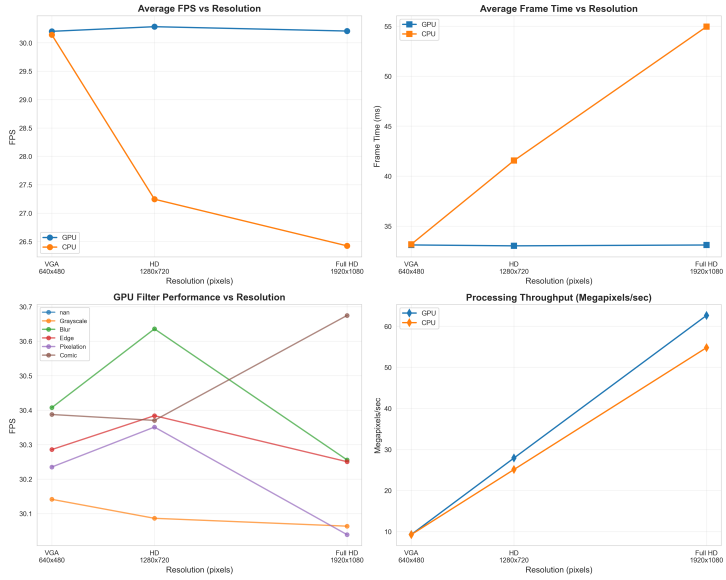


Figure 3: Performance across different resolutions

As resolution increases, CPU performance degrades more significantly than GPU performance. The GPU maintains a relatively stable FPS across resolutions, while the CPU shows a marked decrease, highlighting the advantages of parallel processing for larger data sets.

2.3 Performance Tables

2.3.1 GPU Speedup Factors resolution

Table 1: GPU speedup over CPU for each filter

Filter	Speedup Factor
None	0
Grayscale	0.9911051151304507
Blur	0.9832661231455018
Edge Detection	0.9905437890552572
Pixelation	0.9921230970698969
Comic Art	5.056022571870571

2.3.2 GPU Speedup Factors transforms

Table 2: GPU speedup over CPU for each filter

Filter	Speedup Factor
None	0
Grayscale	0.9925609416695316
Blur	0.9892095840826678
Edge Detection	0.9861874029899519
Pixelation	0.9961871093584084
Comic Art	4.996309606519084

The results show that the GPU only provides a significant speedup for the Comic Art filter, while for all other filters the speedup is negligible or even negative.

3 Analysis and Discussion

3.1 CPU vs GPU Performance

3.1.1 Unexpected GPU Performance

The most striking finding from the benchmarks is that GPU processing shows no performance advantage over CPU processing for most filters, and in some cases performs slightly worse (speedup factors of 0.98-0.99). This was unexpected, some of the factors that could lead to this, are listed below:

- **Memory Transfer Overhead**: Each of the frames are going from system RAM to GPU memory as a texture. Now for the tested resolutions all the way up to 1920×1080, this transfer time dominates the processing time, negating any computational advantages.
- **Simple Filter Complexity**: Most implemented filters (grayscale, blur, edge detection) are relatively simple operations. The computational load is low enough that

the CPU can handle them efficiently without the need for parallel GPU processing. Hardware used is a M4 Pro chip with 16 core CPU.

- **Webcam Framerate Limitation**: The camera operates at approximately 30 FPS maximum, creating a bottleneck that prevents either of the processing methods from showing its full potential.

3.2 Filter Complexity

3.2.1 Filter Performance Characteristics

All filters except Comic Art maintain similar performance (30 FPS on both CPU and GPU), suggesting:

1. The camera capture rate is the primary bottleneck
2. Simple filters don't stress either processing architecture
3. More complex filters (Comic Art) reveal processing architecture differences, which is therefore also the most interesting filter

3.3 Resolution Impact

From the resolution benchmark data (VGA to Full HD, representing a 6.75 times pixel increase):

- **FPS Variation**: Performance drops for the CPU are very evident, compared to the GPU, when upscaling resolution, performance worsens. Like seen in 3. This is because the CPU has to handle more data without the parallel processing capabilities of the GPU.

3.3.1 Implications

The results suggest that for higher resolutions, GPU processing becomes more advantageous due to its ability to handle larger data sets more efficiently through parallelism. This is especially true for more complex filters, where the computational load is higher.

3.4 Transform Overhead

3.4.1 GPU Transform Efficiency

Geometric transformations on the GPU (vertex shader operations) show negligible overhead:

- Transformations are mathematical operations on 4 vertices per frame
- GPU vertex processors handle these operations trivially
- No measurable FPS difference between transformed and non-transformed rendering

3.4.2 CPU Transform Cost

CPU transformations using `warpAffine()` introduce overhead:

- Requires interpolation calculations for every pixel
- Memory access patterns become less cache-friendly
- Additional processing step in the pipeline

However, the overhead remains minimal (1-2% FPS reduction) due to:

- Camera framerate limiting overall throughput
- Efficient OpenCV implementation
- Simple affine transformations (not perspective warps)

3.5 Shader Design and Its Limitations

3.5.1 Shader Architecture

The GPU processing pipeline utilizes GLSL 3.3 with a two-stage architecture:

- **Vertex Shader**: Handles geometric transformations (translation, rotation, scale) on 4 vertices per frame
- **Fragment Shaders**: Six implementations for different filters, executing in parallel across all pixels

The transformation pipeline applies operations in order: scale → rotate → translate, using a 2D rotation matrix:

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{1}$$

Filter Implementations Each filter is implemented as a fragment shader:

1. **Grayscale**: Luminance calculation using perceptual weights (0.299, 0.587, 0.114)
2. **Gaussian Blur**: 5×5 convolution kernel (25 texture samples per pixel)
3. **Edge Detection**: Sobel operator with thresholding at 0.3
4. **Pixelation**: Grid-based coordinate quantization
5. **Comic Art**: Combined edge detection and 4-level color quantization

3.5.2 Key Limitations

Texture Upload Bottleneck The primary performance limitation is CPU-to-GPU data transfer, occurring every frame:

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
             frame_rgb.cols, frame_rgb.rows,
             0, GL_RGB, GL_UNSIGNED_BYTE,
             frame_rgb.data);
```

At higher resolutions like Full HD (1920×1080), several megabytes of data must be transferred from system RAM to GPU memory for each frame. At typical webcam frame rates, this continuous data transfer requires substantial memory bandwidth. The lack of GPU performance advantage for simple filters suggests that this transfer overhead may be comparable to or exceed the time saved by GPU parallel processing, effectively masking any computational benefits.

Synchronization Overhead Each frame requires CPU-GPU synchronization via `glfwSwapBuffers()`, introducing latency and preventing asynchronous operation. CPU-only processing avoids this synchronization overhead entirely.

Memory Bandwidth for Kernel Operations Convolution-based filters require multiple texture samples per pixel:

- Gaussian Blur: 25 samples per pixel (5×5 kernel)
- Edge Detection: 9 samples per pixel (3×3 Sobel kernel)
- Comic Art: Multiple passes with edge detection and color sampling

For high-resolution video at standard webcam frame rates, the cumulative memory bandwidth required for these repeated texture accesses can approach the limits of integrated GPU memory subsystems, potentially causing cache misses and memory stalls that reduce the effectiveness of GPU parallelism.

3.5.3 Why GPU Doesn't Win

Table 3 summarizes the key differences:

Table 3: Critical differences between GPU and CPU implementations

Aspect	**GPU Shaders**	**CPU (OpenCV)**
Data Transfer	Required every frame	Stays in RAM
Synchronization	1-3 ms per frame	None
Parallelism	Massive (GPU cores)	Limited (CPU threads)
Optimization	Generic shaders	SIMD-optimized
Transform Cost	4 vertices only	All pixels

The results show that for simple filters at 30 FPS and Full HD resolution:

- **Transfer time > Processing time**: Data movement dominates computation

- **Camera bottleneck**: 30 FPS ceiling prevents both architectures from showing full capability
- **GPU wins for complexity**: Comic Art filter (5× speedup) combines multiple operations in one pass, reducing transfer overhead

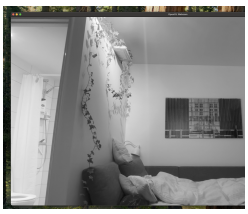
3.5.4 Conclusion

The report discovered that GPU acceleration isn't universally superior:

- Data transfer costs can exceed processing gains for simple operations
- GPU advantages emerge with sufficient computational complexity (Comic Art)
- Graphics rendering pipelines are suboptimal for general image processing
- Camera frame rate limitations mask true performance characteristics

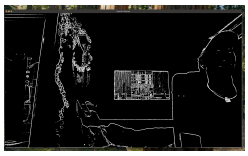
For real-time video processing at typical webcam frame rates, CPU processing with optimized libraries (OpenCV) often matches or exceeds GPU performance, except for complex multi-stage filters.

3.6 Showcasing Filters

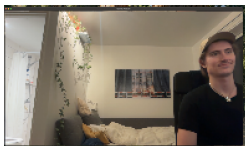


(a) Grayscale

(b) Gaussian Blur (5×5 kernel)



(c) Edge Detection (Sobel)



(d) Pixelation (10×10 blocks)

(e) Comic Art

Figure 4: Visual comparison of implemented filters applied to the same input frame. (a) Original unfiltered frame serves as baseline. (b) Grayscale conversion reduces color information. (c) Gaussian blur creates smoothing effect. (d) Edge detection highlights contours using Sobel operator. (e) Pixelation creates blocky mosaic effect. (f) Comic art combines edge detection with color quantization for artistic effect.