COMPUTER VISION

Real-Time Multi-Camera Systems

How to scale computer vision from 2 to 100+ synchronized cameras using GStreamer and GPU acceleration — hardware selection, synchronization, and pipeline design.

Dr. Farshid Pirahansiah · pirahansiah.com

Real-Time Multi-Camera Vision Systems

Building real-time multi-camera AI requires synchronizing 10–100 cameras while processing with CPU, GPU, NPU, and direct I/O in parallel. This guide covers frameworks (OpenCV, GStreamer, DeepStream, OpenVINO), low-level optimizations (USB DMA, UVC driver tweaks), and scaling strategies.

1. System Architecture

Pipeline

1. **Capture** – RTSP, USB, MIPI-CSI input
2. **Decode** – CPU, GPU hardware decoder, or FPGA
3. **Preprocess** – resize, color convert, normalization
4. **Inference** – DNNs on GPU/NPU/CPU
5. **Post-process** – tracking, feature extraction
6. **Output** – GUI, storage, or network stream

Framework Examples

- **NVIDIA DeepStream**: GPU-accelerated multi-camera inference with batching (nvstreammux), inference (nvinfer), trackers, and OSD
- **Intel OpenVINO**: Multi-Camera Multi-Target demo with detector + re-ID + tracker
- **GStreamer**: Flexible pipelines with hardware decoders, multithreaded elements, and timestamp handling

2. Heterogeneous Processing (CPU/GPU/NPU)

Processor	Role
CPU	I/O, buffering, lightweight pre/post-processing
GPU	High-throughput DNN inference, CUDA/Vulkan processing
NPU/DLA/TPU	Dedicated ML accelerators for extra streams
Multi-threading	Double/triple buffering, async pipelines

Research shows speedup from ~21 FPS (CPU-only) to ~55 FPS with GPU+NPU double buffering — a 2.6x improvement. Beyond 2 NPUs, CPU stages become the bottleneck (Amdahl's Law).

3. Scaling to 10–100+ Cameras

Camera Count	Architecture
< 10	Single GPU system
10–50	Multiple GPUs or edge cluster (Jetson Thor, PCIe grabbers)
50–100	Datacenter with distributed pipeline (Kafka/MQTT)
100+	Server cluster with edge ingestion + cloud aggregation

Key constraints: - Single GPU decodes ~10–20 1080p streams (limited by hardware decoders) - Multi-GPU: each GPU handles a batch, aggregate via messaging - Network bandwidth and I/O are common bottlenecks

4. Software Frameworks

Framework	Strengths	Best For
OpenCV	Easy prototyping, Python/C++, DNN module	Small-scale, custom logic
GStreamer	Flexible pipelines, hardware acceleration, multi-source sync	High-performance, real-time
FFmpeg	Mature video I/O, codec support	Encoding/streaming
NVIDIA DeepStream	GPU/DLA inference, batching, tracking	Jetson/dGPU multi-camera
Intel OpenVINO	CPU+GPU+VPU heterogeneous execution	Intel hardware
OBS + NTP/PTP	Software sync across consumer cameras	Prototyping

5. Camera Synchronization

Hardware Sync (Best)

- **Genlock**: Shared clock signal across cameras (pro cameras only)
- **FSYNC**: Frame start trigger from master sensor
- **PTP (IEEE 1588)**: Network-based precision clock sync

Software Sync (Consumer Cameras)

- **Timestamp alignment**: Each frame tagged with PTS, align within ±1 frame time
- **NTP/PTP network sync**: Align internal clocks across IP cameras
- **Timecode (LTC)**: SMPTE timecode via audio or metadata (libltc)

OpenCV Python Sync Example

```python
import cv2, time, threading
from collections import deque

CAM_IDS = [0, 1]
TOLERANCE_MS = 30

class CamReader(threading.Thread):
    def __init__(self, id, buf):
        super().__init__(daemon=True)
        self.id = id
        self.cap = cv2.VideoCapture(id)
        self.buf = buf
    def run(self):
        while True:
            ret, frame = self.cap.read()
            if not ret:
                time.sleep(0.01)
                continue
            self.buf.append((time.time() * 1000, frame))

def pop_synced(buffers, tol_ms):
    heads = [b[0][0] for b in buffers if b]
    if len(heads) < len(buffers):
        return None
    if max(heads) - min(heads) <= tol_ms:
        return [b.popleft()[1] for b in buffers]
    buffers[heads.index(min(heads))].popleft()
    return None

bufs = [deque() for _ in CAM_IDS]
for i, cid in enumerate(CAM_IDS):
    CamReader(cid, bufs[i]).start()

while True:
    synced = pop_synced(bufs, TOLERANCE_MS)
    if synced:
        for i, frm in enumerate(synced):
            cv2.imshow(f'cam{i}', frm)
        if cv2.waitKey(1) == 27:
            break
```

GStreamer Quick Start

```
# Side-by-side display of two cameras
gst-launch-1.0 videomixer name=mix ! videoconvert ! autovideosink \
  v4l2src device=/dev/video0 ! video/x-raw,width=640,height=480 ! videoconvert ! mix. \
  v4l2src device=/dev/video1 ! video/x-raw,width=640,height=480 ! videoconvert ! mix.

# RTP/UDP camera sync
gst-launch-1.0 -v udpsrc port=5000 caps="application/x-rtp" \
  ! rtph264depay ! h264parse ! avdec_h264 ! videoconvert ! autovideosink
```

6. Low-Level Camera & Buffer Handling

USB Transfer

- USB bulk transfer sends 1024-byte packets
- Camera stream split into packets, reassembled in buffer
- Match USB frame pacing to avoid drops

Buffer Management

- **Skip/lock/unlock strategy**: Single buffer with selective access
- **Triple buffering**: Capture, process, display simultaneously
- **Threading**: One thread per camera, central sync manager

UVC (USB Video Class)

- UVC adds 12-byte headers per packet
- Driver strips headers before passing buffer
- Can bypass for raw access

Direct Memory Access (DMA)

- Stream frames directly into memory without CPU copy
- Direct USB → buffer via DMA
- Zero-copy frame access for GPU/OpenGL/DirectX

Driver-Level Optimizations

- Remove UVC headers early to free CPU
- Use raw capture (no buffering at driver level)
- Enable overwrite mode for DMA buffer

7. Hardware Technologies

Hardware	Camera Capacity	Bandwidth	Use Case
Jetson Thor	20+ CSI/MIPI	>400 Gbps	Robotics, medical AI

Hardware	Camera Capacity	Bandwidth	Use Case
AverMedia T4000 (PCIe)	20 USB/MIPI	PCIe Gen4 ~128 Gbps	Surveillance, multi-view
MIPI C-PHY	1–4 per lane	~17.1 Gbps/lane	High-res industrial/medical
MIPI D-PHY	1–4 per lane	~4.5 Gbps/lane	Smartphones, IoT
QSFP (40/100/200G)	50–100 aggregated	40–200 Gbps	Datacenter clusters
USB 3.0 Hub	4–8 per hub	~5–10 Gbps/controller	Prototyping

Scaling Decision Flow

```
1–4 cameras    → USB 3.0 ports/hubs
5–10 cameras   → Multiple USB hubs + PCIe expansion
10–20 cameras  → PCIe capture cards (T4000) or Jetson Thor
20–50 cameras  → Multiple PCIe grabbers or MIPI C/D-PHY
50–100 cameras → QSFP aggregation + datacenter GPU/NPU
100+ cameras   → Distributed cluster + PTP sync
```

8. Medical Computer Vision

Multi-camera systems in healthcare: - **Surgery**: Multi-angle views to reduce occlusion, staff tracking - **Endoscopy**: Stereo/multi-view depth reconstruction - **ICU monitoring**: Fall detection, movement tracking - **Medical robotics**: Robot navigation with multi-camera fusion

Studies show multiple camera angles are essential in OR settings to overcome instrument/hand occlusion.

9. OCR and Redaction Pipeline

For real-time text detection and sensitive data masking across multiple streams:

1. **Text Detection**: EAST/CRAFT detector on GPU (~250 FPS on V100 vs ~10 FPS on CPU)
2. **PII Classification**: Regex or NLP model to tag sensitive text
3. **Redaction**: Gaussian blur or black bar on detected regions
4. **Frame Output**: Filtered frame to display or stream

Target: ≥15 FPS per camera with GPU-accelerated OCR.

10. Key References

- NVIDIA DeepStream SDK documentation

- Intel OpenVINO Multi-Camera demos
- GStreamer multi-camera pipeline examples
- RidgeRun GstCUDA zero-copy framework
- Oh et al. (2023) — heterogeneous NPU processing benchmarks
- Hu et al. (2022) — OR multi-camera surveillance system