

From GraphEdit to GstPipelineStudio

20 years of fun with multimedia
pipelines



Stéphane Cerveau

Gstreamer conference 2026, Praha 2026

Who am I?



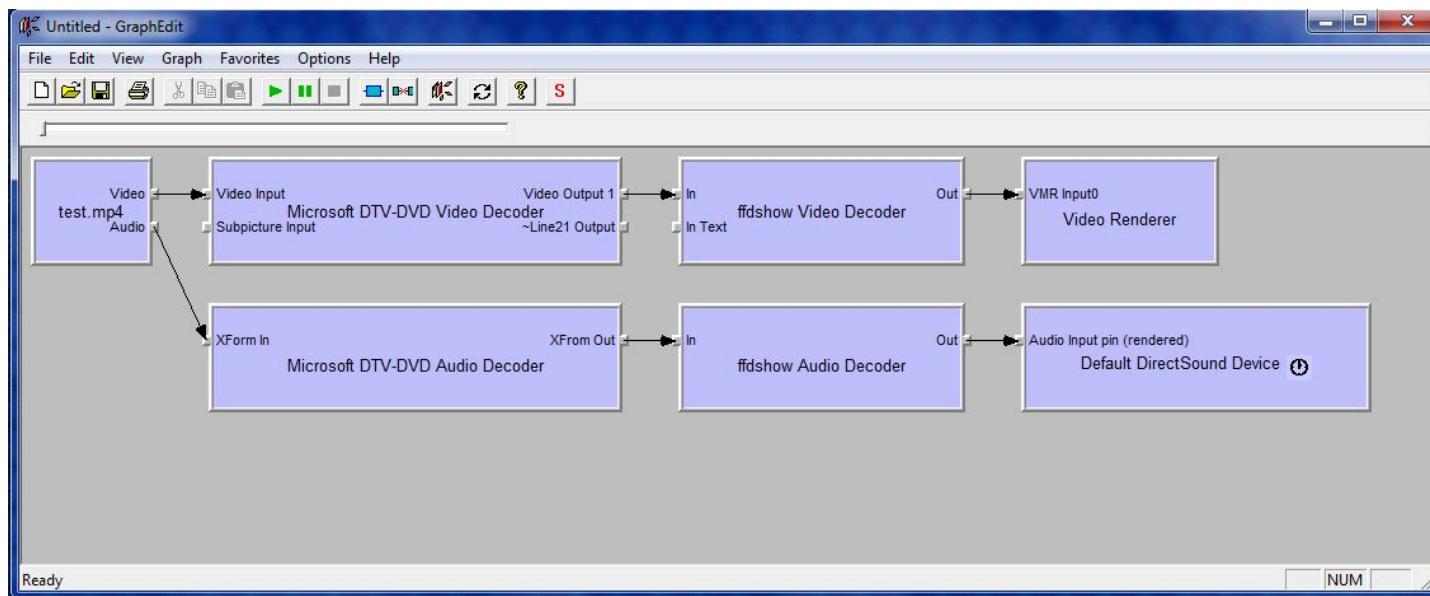
- **Stéphane Cerveau**, based in Spain
- Free Software enthusiast, consultant @ **Igalia**
- **GStreamer developer**: ~600 commits since 2013
- 20+ years of **embedded multimedia**
- mooday.dev · [dabrain34](https://dabrain34.com)

The journey

1. **DirectShow**: filter graphs, COM and the Windows registry
2. **TI DaVinci**: GStreamer 0.10 on ARM + DSP
3. **An audiophile player**: a D-Bus media daemon, AirPlay and friends
4. **MPEG-DASH**: my first real plugin debugging
5. **Dashsink**: my first element upstream
6. **Static GStreamer**: the big per-feature refactoring
7. **Vulkan Video**: debugging on the GPU
8. **The debug toolbox**: the well-known tools, and the hidden gems
9. **GstPipelineStudio**: back to the GraphEdit idea

1. Before GStreamer: DirectShow

Pipelines, the Microsoft way



Same ideas, different names

DirectShow	GStreamer
Filter	Element
Pin	Pad
Media type	Caps
Filter graph	Pipeline
Intelligent connect	<code>decodebin</code> / <code>playbin</code>
Merit	Rank
GraphEdit	<code>gst-launch</code> ... and GPS

- Later I wrapped GStreamer **inside** a DirectShow filter (`dshow-fluendo-gst`): my first [GstConf lightning talk](#) (2018) 🤩



My DVB-T / ATSC player

Digital TV on Windows with the **BDA** (Broadcast Driver Architecture) filters:

```
Network Provider → BDA Tuner → BDA Capture
(DVB-T / ATSC)      (driver)      (driver)
                                     |
MPEG-2 Demultiplexer ←───────────┘
├─▶ BDA MPEG2 TIF (PSI/SI tables)
├─▶ audio decoder → audio renderer
└─▶ video decoder → VMR
```

- One graph, **two worlds**: same filters, different tuning spaces and tables

2. TI DaVinci 2005

GStreamer 0.10 on ARM + DSP

The DaVinci SoC

- **ARM9 + TI C64x+ DSP** on one chip (DM644x, DM365...)
- Codecs run on the DSP through **Codec Engine / DMAI**
- Cant remember of the beautiful kernel module name, can someone help me ?? :)
- GStreamer **0.10**: `video/x-raw-yuv`, `ffmpegcolourspace`, `decodebin2`
- my first C programs to build a pipeline with `gst_element_factory_make`, `gst_bin_add_many` and `gst_element_link_many`

My first pipelines

Decode H.264 on the DSP, render on the TV output:

```
gst-launch filesrc location=sample.264 ! typefind \  
  ! TIViddec2 codecName=h264dec engineName=codecServer \  
  ! TIDmaiVideoSink videoStd=D1_NTSC videoOutput=COMPOSITE sync=f
```

- Lesson 1: **gst-launch** is the best prototyping tool ever
- Lesson 2: GObject was much better than COM interface from Windows.

3. The audiophile player

A decentralized player over D-Bus



cl-mpm: a media daemon

- A connected **audio capsule** (Voxtok), 2015–2016
- No monolithic app: **Daemon talking over D-Bus with ruby on rails server**
 - `cl-mpm`: media player / transcoder / ripper agents
- Any UI (web, mobile, buttons) is just a **D-Bus client through ruby on rails**
- Exposing other interface such **MPRIS2** plus a custom Voxtok interface

The fun features

- **Gapless playback**
- **BPM detection** of your music library: `source ! queue ! decodebin ! audioconvert ! bpmdetect ! fakesink`
- **CD ripper**: `cdparanoiasrc ! ... ! encoder ! muxer ! filesink`
- **Multiroom**: one capsule streams to the others over RTP
- **server**: `... ! audioconvert ! rtpL16pay ! udpsink`
- **client**: `udpsrc ! rtpbin ! rtpjitterbuffer ! rtpL16depay ! alsasink sync=true`
- **Web radios** with `souphttpsrc`, even `mmssrc`
- **AirPlay receiver**: `airplaysrc`, shairport in an element

The D-Bus API

```
<method name='playURI'>
<method name='streamURI'>
<method name='playFromM3UFile'>
<method name='enqueueItemToPlaylist'>
<method name='playSpeed'>
<method name='shuffleMode'>
<method name='setAudioDevice'>
<method name='toggleMute'>
```

- Same idea later in **GstPop**: a GStreamer daemon behind D-Bus / WebSocket

Shipping it: GStreamer in Yocto

- poky recipes: one per module, **one package per plugin** (`gststreamer1.0-plugins-base-playback...`)
- Features through `PACKAGECONFIG`... or a `.bbappend` when disabled hard (`bpmdetect`):

```
# gststreamer1.0-plugins-bad_%.bbappend
PACKAGECONFIG_append = " libmms"
EXTRA_OECONF_remove = "--disable-soundtouch"
EXTRA_OECONF += "--enable-soundtouch"
DEPENDS += "soundtouch"
```

- Own recipes: `cl-mpm` (`inherit cmake`), `airplaysrc` + `libshairport` (`inherit autotools`)

4. MPEG DASH Demux

My first real plugin debugging



MPEG-DASH in 1 slide

- **MPD** manifest (XML): Period → AdaptationSet → Representation → Segments
- The player picks a **representation** and downloads **segments** over HTTP
- Many moving parts: HTTP, manifest parsing, live clock, bitrate switching.
- Need a lot of features to support HBBTV standard.
- GStreamer: downstream plugin different from dashdemux2 using a lot of upstream features.

Debugging: start with the logs

- Reading a lot of logs using `less -RS`
- Viewing pipeline using dots generation
`GST_DEBUG_DUMP_DOT_DIR`
- List every debug category: `gst-launch-1.0 --gst-debug-help`
- display message/events/tags etc.
- multithreading with queue debugging.

Debugging: crashes and leaks

```
# Stop on the first g_critical / g_warning
G_DEBUG=fatal-criticals gdb --args gst-launch-1.0 playbin3 uri=$U
(gdb) run
(gdb) thread apply all bt

# Who leaks what?
GST_TRACERS=leaks GST_DEBUG=GST_TRACER:7 gst-launch-1.0 playbin3

# Latency per element
GST_TRACERS="latency(flags=element)" GST_DEBUG=GST_TRACER:7 gst-l
```

5. dashsink

My first element upstream

From reading to writing MPDs

- 2019: **split** the monolithic `gstmpdparser.c` into `mpdparser` / `mpdclient` / `xmlhelper`
- Parser nodes become **GObjects** with properties
- Add XML **generation** from those objects
- Goal: reuse the same MPD model to **write** manifests...

dashsink

- Written in 2019, merged January 2020, released in **GStreamer 1.18**
- My first full GStreamer Conference talk:
[dashsink plugin to complete the GStreamer offering](#) (Lyon, 2019)
- The mpdclient **writes the MPD** (static or **dynamic** live)

```
gst-launch-1.0 dashsink name=dashsink mpd-root-path=/var/www/dash  
    target-duration=4 dynamic=true \  
    audiotestsrc is-live=true ! avenc_aac ! dashsink.audio_0 \  
    videotestsrc is-live=true ! x264enc ! dashsink.video_0
```

- Later: **dashsink2** support (2025)

6. The big refactoring

A static, minimal GStreamer



gststreamer-full

- One library with **GStreamer** + **all the plugins**, for embedded and mobile
- `-Dgst-full-target-type=static_library` (or shared)
- Version script, `-Bsymbolic-functions`, unit tests...
- **Problem**: linking a plugin statically pulls **all its elements**. Need `filesrc`? You get the whole `coreelements`.

```
meson setup build -Dgst-full-target-type=static_library \  
  -Dgst-full-plugins='coreelements;playback;typefindfunctions' \  
  -Dgst-full-elements='coreelements:filesrc,fakesink,identity'
```

The cost of touching everything

- ~**180 commits** "allow per feature registration", nearly every plugin
- 2021: my busiest year upstream (**233 commits**), a lot due to this change to be fair ... :)
- The fun part, the bugs that came after:
 - `rtp: missing debug init after element splitting`
 - `autodetect: fix debug init category`

but not that much at end ... 😅

7. Vulkan Video

Debugging on the GPU



Vulkan Video in GStreamer

- Decoders: **H.264, H.265, AV1, VP9** · Encoders: **H.264, H.265, AV1**
- One API, many drivers: **ANV, RADV, NVIDIA**, and two platforms, Windows and Linux
- The bug can be anywhere: **GStreamer ↔ Vulkan loader ↔ Mesa driver ↔ firmware**
- Work across the stack: GStreamer, Mesa, CTS, Fluster, FFMpeg

The Vulkan Video debugging toolbox

```
# Validation layers catch API misuse before the driver does
VK_INSTANCE_LAYERS=VK_LAYER_KHRONOS_validation \
GST_DEBUG=vulkanh264dec:6,vulkandecoder:6 \
  gst-launch-1.0 filesrc location=in.mp4 ! parsebin \
    ! vulkanh264dec ! vulkandownload ! fakesink

# Conformance: is it my element or the driver?
fluster.py run -d GStreamer-H.264-Vulkan -ts JVT-AVC_V1
```

- Some contributions along the way:
 - **vulkanvp9dec**: VP9 decode support and element (1.28)
 - **AV1 encode** support in the Vulkan encoder library
 - H.265 **Main 10** decode, H.264 **extended profile** & interlaced
 - One element **per GPU** when several devices are present
 - **VK_KHR_video_maintenance1**, H.264/H.265/AV1/VP9 unit

8. The toolbox

The well-known tools, and the hidden gems

The everyday tools

Tool	What for
<code>gst-launch-1.0 -v -m</code>	Prototype, print negotiated caps (<code>-v</code>) and messages (<code>-m</code>)
<code>gst-inspect-1.0</code>	Pads, caps, properties, rank of an element
<code>gst-discoverer-1.0 -v</code>	Streams, codecs, tags of a media
<code>gst-play-1.0</code>	Keyboard player: seek, rate, trick modes, tracks

Hidden gem: playback-test

A GTK test player in `gst-plugins-base/tests/examples/playback/`

```
# Built with the examples (needs GTK 3)
./build/subprojects/gst-plugins-base/tests/examples/playback/play
  playbin file:///path/to/video.mkv
# or any launch line
playback-test parse-launch "filesrc location=in.mp4 ! qtdemux ! h
  ! vulkanh264dec ! vulkandownload ! videoconvert ! autovideosi
```

- Seek slider: **accurate** / **key-unit** / **flush** / **snap before** / **snap after**
- **Rate from -100 to 100**: fast forward and **reverse playback**
- **Frame stepping**, scrubbing, instant rate change, frame artifacts
- Trick modes: **keyframes only**, no audio

Automate it: gst-validate

```
# Record a reference description of a media...
gst-validate-media-check-1.0 file:///path/in.mp4 --full -o in.medi
# ...and compare against it after your decoder change
gst-validate-media-check-1.0 file:///path/in.mp4 --full -e in.medi

# Replay the playback-test clicks as scenarios
gst-validate-1.0 --set-scenario=reverse_playback playbin uri=file
gst-validate-1.0 --set-scenario=seek_forward_backward playbin uri
```

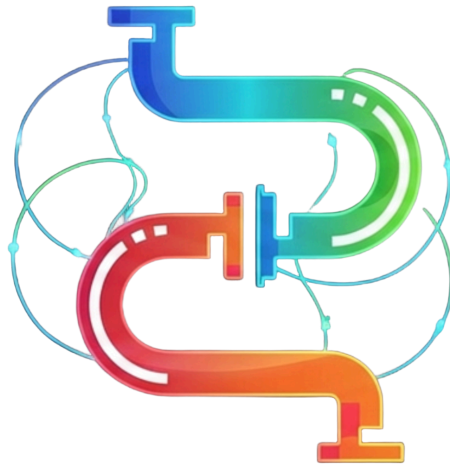
- Scenarios also exist for `fast_backward`,
`scrub_forward_seeking`, `trick_mode_seeks`,
`switch_audio_track`...

More hidden gems

- **gst-debug-viewer** (gst-devtools): filter and color huge GST_DEBUG logs
- **gst-stats-1.0**: summary of a tracer log (buffers, events, per pad/element)
- **pipeline-snapshot tracer** (gst-plugins-rs): DOT files on demand, without touching the app
- **YUVview tool**: Oops! this one is not using GStreamer but really useful (I have a dream....)

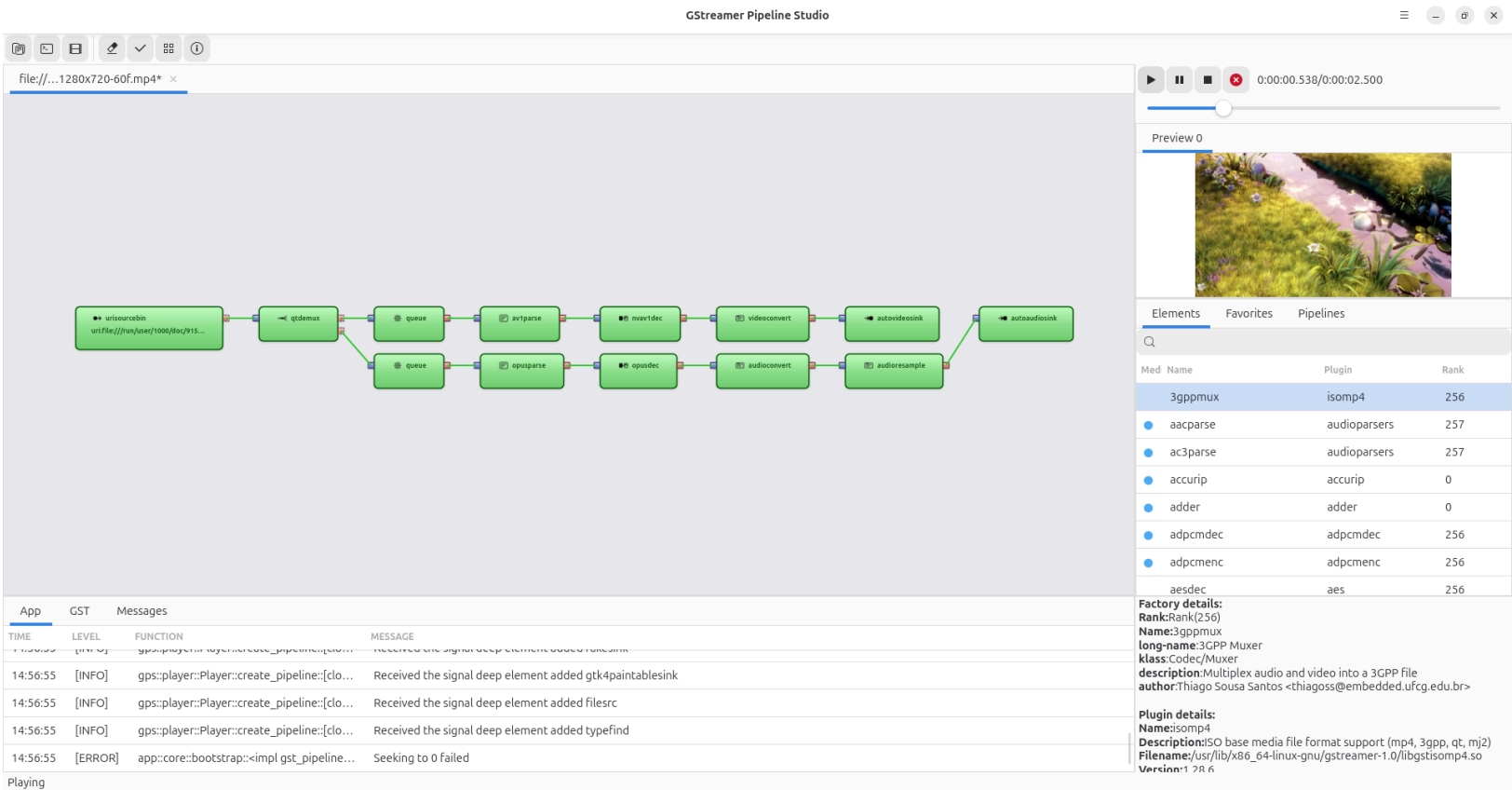
9. GstPipelineStudio

Back to the GraphEdit idea



A GUI for GStreamer

- Draw your pipeline and play it
- Debug it: test elements, filter logs, **open .dot files**, connect to running pipelines, read tracer stats
- Rust + GTK4, Linux / Windows / macOS
- Everything I needed during these 20 years, in one tool



What's new in 0.6.x

- **Caps-based auto-suggest**: click a src pad, get the compatible next elements
- **Open Media**: file or URL (DASH / HLS too) → pipeline built for you
- **Element swap**: choose another compatible element in place
- **GstPop** ↔ **GPS**: snapshot a *remote live* pipeline as DOT, render it in GPS,
- **Logs**: Support high load with detached log handler
- **stats**: display buffers, events, tags, queries live...

Full circle

Then	Now
GraphEdit	GstPipelineStudio
<code>gst-launch</code> on DaVinci	Auto-suggest from negotiated caps
cl-mpm over D-Bus	GstPop over D-Bus / WebSocket
<code>dot -Tsvg</code> on DASH dumps	Open Dot Folder in GPS

Lessons from 20 years

- The pipeline model has survived every framework, OS and hardware change
- **Logs, DOT files and gdb** solve most problems
- Upstream your work: your hacks become everyone's features
- Build the tools you wish you had

Useful information

- [GStreamer documentation: debugging tools](#)
- [GstPipelineStudio: project page](#)
- [GstPipelineStudio: source](#)
- [GstPop / GstPrinceOfParser](#)

Thanks

Q/A

