

Video Player Internals

BOSSA 2008, Porto de Galinhas

Conrad Parker

March 22, 2008

1 Introduction

1.1 Abstract

Video Player Internals

Embedded platforms put demands on latency and memory use. Video playback makes these difficult to guarantee. This presentation discusses the architecture of video players, and the problems imposed on them by the design of video codecs and their containers. To explain these problems we look at both proprietary and open source formats (MPEG, Ogg, Theora, Dirac, etc.) and evaluate open source video players in this context. We particularly examine xine and GStreamer, and introduce the minimal architecture of OggPlay.

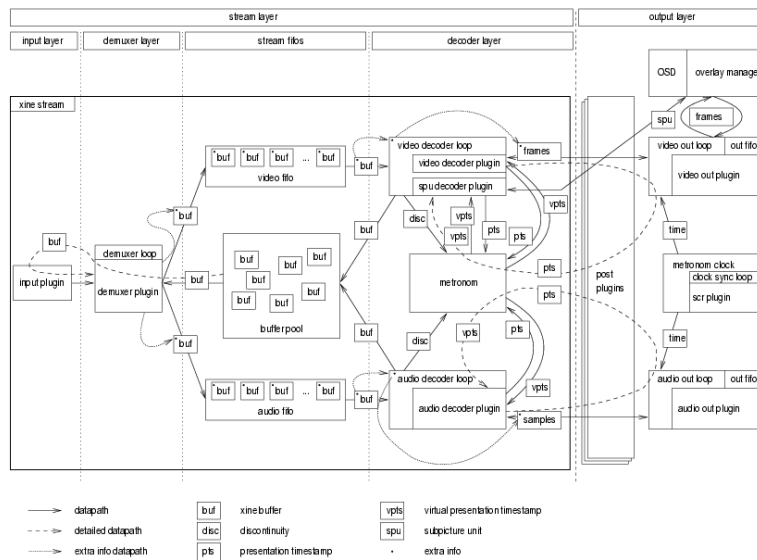
1.2 Outline

Outline of topics

- Goals
- Assumptions
- Player architecture
- Failure modes
- Codec issues
- Container issues

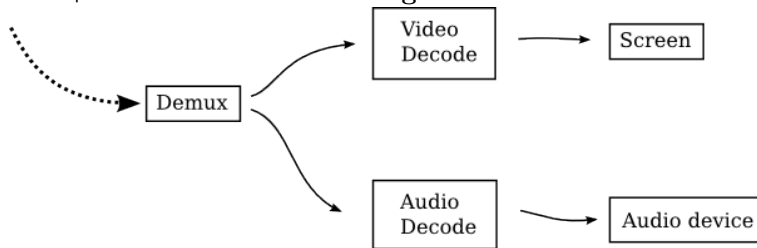
1.3 xine architecture

xine architecture



1.4 Demux+decode

Demux+decode architecture diagram



2 Goals

2.1 Synchronization

Goal: Synchronize audio and video

The general aim is simply to present audio and video in sync.

- We say that audio and video are in sync when audio is heard at the same time as the appearance of the visual image representing it. Think of a clapper board in a film production.

Doing so involves receiving data from the network or disk, separating it into audio and video frames, interpreting and adjusting their timestamps, re-ordering and scheduling when to decode them, and finally rendering these to the respective devices, allowing for variability in rendering time.

2.2 Latency

Goal: Minimize latency

The aim is to minimize the latency between data arriving from the network and being rendered. This is especially important for two-way communications.

- Compressed audio and video take CPU to decode
- The time taken to decode each frame in a stream is not constant
- Audio and video frames may arrive in bursts and out of order

Video players use memory buffers to correct frame ordering, and to smooth the variance in decoding time of each frame. Since you cannot really drop audio frames, but can do so for video, you should give decoding priority to audio over video.

2.3 Memory

Goal: Minimize memory use

On a small device, we also aim to minimize memory requirements, and may need to place an upper bound on buffer allocations. Allocations incur overhead, and on a DSP implementation it may not be possible frame-by-frame (ie. need to know max allocation upfront).

- Network buffer (like on YouTube).
- Encoded data: demux incoming data, prepare timestamps and re-ordering.
- Decoded data buffers: ready to play, but large.

Buffering encoded data requires less memory than buffering decoded data, BUT increases the indeterminism of the amount of memory needed.

3 Assumptions

3.1 Network

Assumption: Bandwidth is non-constant

The first assumption is imposed by TCP/IP networks. IP is a best-effort protocol: other applications on the device, and other users on the network, can affect the amount of bandwidth available at any instant in time.

- The situation is different in telephony and digital television. As a result, earlier standards for video (like MPEG-2) were designed for constant bandwidth.

3.2 Video

Assumption: Video rendering is instantaneous

We build a video image in memory as a block of pixels. We may write directly to special memory which is then accessed directly by the video hardware for rendering, or the video device may blit the image from main memory to screen. Either way, we assume that rendering a frame incurs a negligible, constant delay. It is under the control of our software to manage the timing of rendering, so maintenance of the image framerate is derived from the CPU clock.

- For video, minor timing errors up to tens of milliseconds are not noticeable.

3.3 Audio

Assumption: Audio rendering is buffered in the device

The audio hardware converts digital audio (PCM samples) into an analog electrical signal which the speakers then convert into sound. It must do this constantly or else the sound produced becomes incoherent. So it has its own clock, and continually converts into sound whatever digital samples it has available.

- For audio, timing errors of tens of milliseconds are very noticeable, so the audio device must have an uninterrupted stream of available samples.

In general, application software running on the CPU cannot guarantee to supply samples on time, so the audio device keeps an oversupply of buffered audio. An application writes to a circular buffer, and the audio device reads from it.

3.4 Clocks

Assumption: Clocks are not perfect

A clock is a piece of hardware which provides timing, typically a crystal oscillator in an integrated circuit. These are imperfect devices; a clock may vary slightly with age or temperature, and any two different crystals are out of time by some small amount. The difference in timing between two clocks is called **drift**.

- Generally, the CPU and the audio device each have their own clock.
- The timestamps in a video stream refer to the clock of the device that the content was encoded on. The audio and image tracks may even have been encoded on different devices.

4 Player architecture

4.1 Player architecture

Player architecture

- OggPlay
- xine
- GStreamer
- Maemo

4.2 OggPlay

OggPlay

OggPlay is a lightweight playback scheduling library for Ogg files. It is being incorporated into a future Mozilla Firefox 3.x release for HTML5 video support.

- An application using liboggplay receives a callback for each frame of decoded video, and all the audio for that frame.
- Ogg contains no data skew hints to handle, and handling device drift is up to the application.

4.3 xine

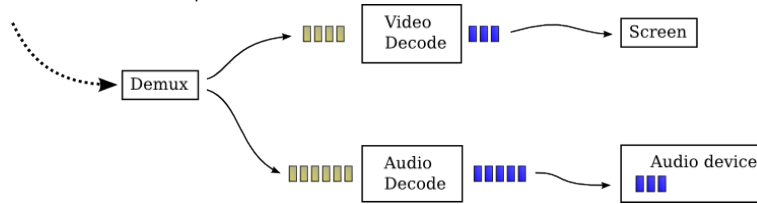
xine

xine uses a single demux, decode pipeline architecture with chained post-processing filters.

Data buffers are tagged with **vpts**:

- **vpts: virtual presentation timestamps**: textttpts in MPEG streams may wrap, be missing or inaccurate.

Buffered demux+decode



4.4 xine clocks

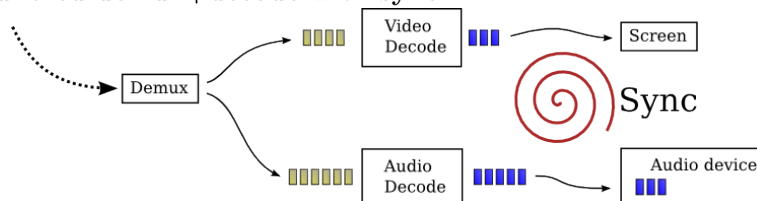
xine clocks

Metronom accounts for wraparound (detects and applies delta) and smooths drift.

xine uses the system clock as a master, but it is possible for hardware-specific plugins to provide a different clock reference. Drift from the soundcard clock can be handled in one of two configurable ways:

- Drift can be fed back into *metronom*, which then adjusts the global vpts. However the audio fifo adds a delay to this closed loop.
- The audio can be stretched or squeezed a slight bit by resampling.

Buffered demux+decode with sync



4.5 GStreamer

GStreamer

GStreamer is a library which allows applications to build an arbitrary graph of demux, decode, effects, encode, mux and I/O elements. Hence it is useful for much more than just playback; you can use it to access the camera, do two-way communications etc.

4.6 Maemo

Maemo

Maemo's multimedia architecture is GStreamer based, and can use the on-board DSP for decoding of some formats. Demux and synchronization is handled by the CPU.

4.7 GStreamer clocks

GStreamer clocks

GStreamer uses a global clock to synchronize the plugins in a pipeline.

Every buffer in a GStreamer pipeline has a `timestamp`, a `duration`, an `offset`, and an `offset_end`.

Renderers synchronize to the global clock using the buffer timestamps. The master clock is usually based on the primary audio device.

Applications can build pipelines which handle multiple clocks, for dealing with multiple soundcards, cameras, and network streams. In a pipeline, clocks can be slaved and calibrated off each other.

5 Failure modes

5.1 FAIL: Jerkiness

FAIL : Jerky playback

Artifact: Jerky playback

Cause: Network jitter (TCP/IP) This is the kind of artifact you are familiar with from YouTube, when you are low on bandwidth.

Fix : Generous anticipation of required network buffers, though this immediately introduces latency. With TCP/IP, about twice the stream bandwidth is required to minimize jitter.

5.2 FAIL: Lost sync (1)

FAIL : Lost synchronization

Artifact: Audio and video appear out of sync

Cause: Clock drift: Failure to correctly account for device clocking.

Fix : Poll devices for their clock times and adjust rendered data to compensate.

5.3 FAIL: Dropped frames

FAIL : Dropped frames

Artifact: Video frames lost, short freezes

Cause: Usually not enough processing power to decode video frames in time for presentation, but sometimes a scheduling issue.

Fix :

- If the device theoretically has enough processing power to decode the images, this may be caused by bad scheduling: for example, too much future audio may have already been decoded at the expense of images.

Alternatively, the decode buffers may be too short to allow for the time required to decode large keyframes.

- Fix the decode scheduling, use a faster CPU or offload decoding to a DSP.

5.4 FAIL: Underruns

FAIL : Playback underruns

Artifact: Choppy, stuttered audio

Cause: Failure to write audio data to the circular playback buffers in the audio device quickly enough, so the D/A converter replays earlier data.

Fix : Ensure that there is always data in the playback buffers, by:

- Using larger playback buffers to minimize errors due to scheduling.
- Use realtime scheduling to ensure the application has the opportunity to write data as needed.

6 Codec issues

6.1 Codecs

Influence of codec features on decoding

Compressed audio and video generally comes in blocks which must be decoded in their entirety, so the time taken to decode a block adds a delay in the player pipeline.

Modern codecs are variable bitrate, making it difficult to correlate the amount of incoming and decoded data. This can compound the problems of unreliable network bandwidth.

We will consider features of various families of codecs which further influence the decoding delay and memory usage.

6.2 Low latency codecs

Low latency: Raw data, VoIP

If each packet of encoded data can be decoded as soon as it arrives, then the decoder adds no latency. The most obvious case of this is when playing raw data such as PCM audio from CD. Some codecs like Speex also have this property, which is useful for low-latency applications like VoIP.

6.3 Preroll

Constant latency: Preroll

Preroll (in a codec) refers to the use of encoded data from a fixed number of preceding packets. This number of packets must first be given to the decoder before any data is output. This introduces a fixed latency between encoding and decoding.

- For example to decode samples from any packet of Vorbis audio, the previous two encoded packets are also required. This is because the encoding process operates on overlapping blocks of samples to minimize MDCT

encoding artifacts at block edges. For Vorbis, we say that MDCT overlap-add introduces 2 packets of preroll.

6.4 Predictive frames

Low latency: Predictive frames only



I: Intra-frame (Keyframe)

P: Predictive frame: only encode the difference from an earlier frame

- Theora: to decode an arbitrary frame, all video data from the previous keyframe is required.
- MPEG-2 simple profile: no backward or interpolated prediction, so no picture re-ordering is required. This profile is suitable for low-delay applications like videoconferencing.

6.5 Bi-directional predictive frames

Bi-directional predictive frames

B: Bi-directional predictive frames encode the difference from I or P frames before and after.

Encoded frames are re-ordered by dependency, so that the following group of pictures, in display order:



is encoded in dependency order:



- The most common example of this is MPEG-2 main profile.

B-frames decrease encoded frame size, but add about 120ms decode delay to allow for picture re-ordering.

6.6 B-Pyramid

B-Pyramid

B-Pyramid refers to a special form of bidirectional prediction, where B-frames may also be used as references. Again, encoded data is stored in dependency order.

- MPEG-4 AVC is the most typical example.
- Dirac can also use a B-Pyramid scheme. For compression efficiency, any frame in the Access Unit may be used as a reference. This makes it difficult to put an upper bound on re-ordering delay.

6.7 FAIL: Dropped reference frames

FAIL : Dropped reference frames

Artifact: Many video frames lost, blocking artifacts.

Cause: Long runs of video frames being dropped is symptomatic of a problem with decoding reference frames. A corrupted reference frame will make it impossible to decode all frames that refer to it. Dropping keyframes is much more problematic than dropping P or B frames. Dropping any frame in a B-Pyramid can be a problem. As with the general case of dropping video frames, the cause could simply be that there is not enough processing power available and scheduled.

Fix : As for the general case of dropping frames, but with extra paranoia to ensure that keyframes can be decoded.

6.8 GStreamer states

GStreamer states

GStreamer elements are always in one of the following states:

- **READY**: resources allocated
- **PAUSED**: frames queued
- **PLAYING**: clock running

7 Container issues

7.1 Timestamps

Issue: Timestamp accuracy

Containers provide timestamps. They record the presentation time for each packet of encoded audio and video data, and ideally record any drift in encoder clocking.

We will consider the following containers:

- Ogg
- NUT
- MPEG

7.2 Ogg

Ogg

An Ogg file consists of independent streams for audio, video, subtitles etc., making it easy to rip apart and to remux. Each page of encoded data contains an absolute timestamp, but there is no inter-stream synchronization.

- The timestamps is encoded in a scheme called *granulepos*. This is lossy, in that not all timestamps submitted to the encoder are represented in the stream; the majority are simply thrown away.

An Ogg page contains many packets, and the *granulepos* recorded in the page represents the presentation time of the last presentable element in the last packet ending on that page.

7.3 FAIL: Lost sync (2)

FAIL : Lost synchronization

Artifact: Audio and video appear out of sync

Cause: Failure to account for drift in encoded audio and video timestamps.

Recall that the audio and video tracks may have been encoded on different devices, with timestamps referencing different physical clocks.

Fix :

- Use a container which encodes drift hints.
- Use encoded drift hints if available in the data.

7.4 NUT

NUT

NUT has explicit `pts` and `dts`, as well as hard sync points (`transmit_ts`).

- **pts: presentation time** of the first frame/sample that is completed by decoding the coded frame
- **dts: decode timestamp** is the timestamp of the first sample output by a decoder when it is fed with the frame. Note that this may be data from previous frames.

NUT also explicitly records `decode_delay`, the size of the re-ordering buffer used to convert `pts` to `dts`. (1 for B-frame, 2 for B-pyramid, usually 0 otherwise).

7.5 MPEG

MPEG

MPEG Systems stream (in MPEG2) and sync layer (in MPEG4) provides time stamps and clock references.

MPEG defines ways for detecting clock drift between sender and receiver, calculating a drift metric and resampling the data in transmission to compensate.

The MPEG timing model also includes ways of synchronizing sender and receiver clocks for realtime transmission, and defines a constant delay from the output of the encoder to the input of the decoder, even for variable-delay channels.

7.6 Muxing overruns

Issue: Muxing can introduce data overruns

Blocks of audio and video have different durations, so they necessarily run ahead of each other. In a container like Ogg, many data packets from the same codec can be bundled together into the same Ogg page. Data overruns become a problem if, for example, we attempt to decode all available video data even though this is many frames ahead of the available audio data. Hence we waste CPU cycles and may not be able to render the audio in time, losing sync.

7.7 oggplay-info

Measuring data overruns with oggplay-info

oggplay-info is a tool distributed with the liboggplay source. It uses oggplay's playback scheduler to measure data overruns in the encoded audio and video data.

Theora: Track 0

Worst overrun: 45 frames

Average overrun: 15.811 frames

Histogram bucket size: 2.250

Histogram: 5 10 10 9 12 10 9 5 6 4 4 3 3 2 2 2 3 2 2 2 1

SD of overrun: 11.357817

8 Conclusion

Conclusion

The basic operations of video playback are demux and audio+video decode, and the goal is to present these in sync, despite delays and drift.

Ultimately the available hardware dictates what is possible to decode. No amount of player re-architecture can reduce the math workload, but a carefully designed player architecture can ensure that the CPU use is scheduled among these tasks just-in-time, avoiding failure and minimizing memory use.

- Conrad Parker conrad@metadecks.org