Что намерено: полный свод

Содержание

> **Про интерфейс в этом документе.** Точки реза здесь находят в плеере mpv, который открывается кнопкой из раздела, и переносят в поля кнопками — либо вводят таймкодом с клавиатуры. Форма волны и прослушивание кадра, о которых говорится ниже, относятся к панели, отложенной до отдельного этапа: движок и измерения от неё не зависят и остаются в силе.

Рабочая тетрадь всего, что меряли по точности резки, синхрону и пределам FFmpeg. Отсюда берутся числа для таблиц отказов в коде и для решений в DECISIONS.md; тот же документ лежит в `E:\TOOLS\fps-test` рядом с разбором Shutter Encoder, откуда эта работа началась.

Правило одно: **ни одно утверждение здесь не взято из документации.** Каждая ячейка таблицы — реальная попытка, каждое число — замер. Где измерение оказалось неверным, это тоже записано: в разделе 11 собраны случаи, когда правдоподобный вывод пришлось отменить.

Версия FFmpeg: **9.0.1**; ключевые числа переснимались на 8.0.1 и совпадают, расхождения отмечены отдельно. В поставку версия попадает не отсюда — её выбирает установщик по версии драйвера NVIDIA, и для большинства машин это ветка 8.x. Поэтому проверять приходится обе.

Порядок величин, за которые идёт разговор: **кадр** (33 мс на 29,97), **звуковой пакет** (8 мс), **сэмпл** (20 мкс). Смещение на один кадр в начале смены — это рассинхрон, который монтажёр будет искать руками.

---

1. Кадр: сколько его в куске

Просили отрезок — сколько кадров вышло. Считается точной дробью, а не десятичной частотой.

Частота	Дробь	Запрос	Правильный ответ	Через `round(fps)`
23,976	24000/1001	50 с	**1199**	1200
29,97	30000/1001	4 мин	**7193**	7200
59,94	60000/1001	4 мин	**14386**	14400
59,94	60000/1001	50 с	**2998**	3000
25	25/1	4 мин	6000	6000

Разница между колонками и есть ×1.001: **3,9 с на час**.

Две детали, на которых легко ошибиться повторно:

- **Точка во времени и таймкод — разные величины.** Non-drop `00:04:00:00` на 29,97 — это кадр 7200, то есть 240,24 с реального времени.
- **Конец отрезка округляется вверх.** Кадр 899 на 29,97 стоит на 29,9966 с — внутри тридцатой секунды, значит он в куске. Округление к ближайшему теряет его, и рез выходит короче на

кадр. Так же и с 2998 кадрами выше: кадр 2997 стоит на 49,99983 с, то есть внутри пятидесятой секунды.

Проверено на материале

Источник	Запрос	Кадров	Расхождение
ProRes 25p	3 → 9	150	0
HEVC long-GOP 29,97	12 → 30	600	0 (после добора, см. раздел 3)
H.264 all-intra 23,976	3 → 9	144	0
29,97 drop-frame	5 → 15	330	0
59,94 drop-frame	3 → 9	360	0
MPEG-TS (старт 1,44 с)	2 → 8	300	0
4 канала PCM	2 → 8	150	0
Canon HEVC 4:2:2 10 бит	3 → 10	166	0
ARRI ProRes 4444 XQ 24,000	2 → 8	144	0
VFR (30 → 17,6)	4 → 16	по времени	+1,07 с, объявлено

2. Частоту не надо угадывать

Соблазн, которому мы поддались и от которого отказались: «камеры пишут круглое там, где идёт дробное, значит 24 надо читать как 23,976». Проверка показала, что `ffprobe` **не округляет**:

Файл	`r_frame_rate`	Измерено (кадры / длительность)
Canon MVI_0757	`24000/1001`	23,97679
Тест-клип 23,976	`24000/1001`	23,97614
ARRI Alexa Mini	`24/1`	**24,00000** (260 кадров на 10,8333 с)

Alexa снимает любую частоту, и снятое на честные 24,000 объявляет `24/1`. Читать такой файл как 23,976 — это создать ошибку ×1.001, а не устранить её: пять кадров за четыре минуты, ровно

то же, за что критикуется чужая нарезка.

Вывод: частота берётся у файла как есть. Ту, что не удалось прочитать, честнее отклонить, чем заменить догадкой.

---

3. Пакет — не кадр

`-count_packets` быстр именно потому, что не декодирует: 0,14 с против 5,7 с на четырёх минутах 59,94. Обычно одно равно другому — но не на хвосте long-GOP.

Что считали	**HEVC long-GOP, запрос 600 кадров**
Пакетов записано	**600**
Кадров декодируется	**597**
Длительность потока	19,920 с вместо 20,020

Последние три B-кадра ссылаются на картинку за границей реза, декодер их отбрасывает. Файл короче на **0,1 с**, при этом все проверки рапортуют успех. То же самое на 8.0.1 — значит это не регрессия версии, а слепое пятно метода.

Недобор устойчив и линеен:

Просим пакетов	**600**	**601**	**602**	**603**	**604**
Декодируется кадров	597	598	599	**600**	601

Выдаёт проблему `duration` потока — она уже считает по-честному и достаётся даром. Поэтому декодировать стоит только тогда, когда длительность не сошлась с планом больше чем на полкадра.

3.1. Превью: где уходит время

Раздел меняет не только рез, но и собственную отзывчивость: фейдер, за которым картинка появляется через минуту, работать не даёт.

Поиск ключевых кадров декодировал картинки. `-skip_frame nokey` пропускает не-ключевые кадры — а на all-intra материале пропускать нечего, и ffprobe честно декодирует всё

окно. На ARRI ProRes 4444 XQ (2944×2160, 12 бит) это **54 секунды** на каждое движение фейдера, при том что сам кадр строится за 0,85 с. Пакеты несут флаг `K` и декодера не требуют:

Способ	ARRI ProRes 15 с окна	ProRes 1080p	HEVC long-GOP
`-skip_frame nokey` (декодирует)	55,2 с	0,31 с	0,05 с
`-show_packets` + флаг K	**0,76 с**	**0,05 с**	**0,04 с**

Списки при этом **совпадают полностью**: проверено на десяти источниках — long-GOP, all-intra, VFR, MPEG-TS, drop-frame, ProRes, ARRI, — худшее расхождение по времени 0,0. Это существенно: от того же списка зависит точка реза.

Масштабирование ничего не стоит; стоит декодирование. Один кадр 4K H.264 из файла на 825 МБ:

Что	Время
Запуск процесса ffmpeg	0,02 с
Кадр **без** уменьшения (4K на выходе)	1,23 с
Кадр **с** уменьшением до 480×270	1,18 с
Ключевые кадры в окне (пакетами)	0,45 с
Волна окна	0,12 с

То есть «вбить 4K в телевизор» — бесплатно, платится за то, чтобы этот кадр добыть: открыть файл, прыгнуть в место и развернуть кадр 3840×2160.

Аппаратное декодирование помогает не всем.

Кодек	Программно	`-hwaccel auto`
H.264 4K	0,85 с	**0,48 с**
HEVC	—	0,35 с
ARRI ProRes 4444 XQ	0,53 с	**пустой кадр**

Аппаратного пути для ProRes не существует ни у одного производителя, и попытка его просить ломает цветовой обходной путь, которым превью этого кодека только и держится. Поэтому ускорение предлагается ровно тем кодекам, которые QSV, NVDEC и AMF действительно

декодируют — список снят с самой сборки (`ffmpeg -decoders`): h264, hevc, av1, vp9, vp8, vvc, vc1, mpeg1/2/4, mjpeg. Монтажные кодеки — ProRes, DNxHR, FFV1, v210 — в нём отсутствуют намеренно. Любая цепочка заканчивается обычным программным декодированием, так что машина без ускорителя картинку всё равно получает.

Итог на слабом ноутбуке (Intel Iris Xe, без дискретной карты):

Материал	Было	Стало
ARRI ProRes 4444 XQ, 1,4 ГБ	~60 с	0,74 с
4K H.264, 825 МБ	—	~1,0 с
HEVC 1080p	—	0,35 с

Звук на превью. Волна показывает, где звук есть, но не что в нём сказано, а раздел существует ради поиска начала фразы. Точку можно послушать: `ffplay -nodisp -autoexit -vn` от текущей позиции. Задержка до первого звука на 825-мегабайтном 4K — около **2 секунд**, и она не зависит от `-probesize`, `-analyzeduration` и `-fflags nobuffer` (проверено, все три дают те же 2,0 с). `-vn` обязателен: декодировать 4K-картинку, которую никто не увидит, значит опоздать с первой нотой.

Из RAW-обёрток звук играется даже там, где картинки нет: у Sony X-OCN и ARRIRAW видео не читается, а дорожки `pcm_s24le` слышны нормально.

Фейдер должен звучать. Найти слово, вдох или паузу, глядя на форму волны, — гадание; в монтажных программах голова звучит, пока её тянут. Проигрывателем это делать нельзя: ffplay стартует 0,35 с на маленьком файле и 1,7 с на 4K, и это процесс на каждое движение.

Звук отдаётся прямо в панель куском WAV. Что это стоит:

Шаг	Время
Вырезать 1 с звука из 4K-файла	0,11 с
Вся дорожка 55-секундного 4K в память	0,16 с (2,5 МБ)
Вся дорожка получасового ролика	1,97 с (82 МБ)
Первый кусок под курсором (пока трек грузится)	0,21 с
Кусок в **любой** точке готового трека	**0,84 мс**

Шаг	Время
Обновление кадра превью (4K, было 0,92 с)	0,69 с

Порядок оказался важнее скорости: сначала звук, потом картинка. Пока раздел рисовал кадр (0,69 с) и перерисовывал волну (0,13 с) прежде чем дать звук (0,8 мс), поиск фразы на слух означал ожидание секунды ради глаз. Теперь кусок звука уходит через 0,12 с после остановки руки, а кадр догоняет своим темпом; окно волны перерисовывается, только когда точка из него вышла. Оттуда же ушли лишние 114 мс — раздел спрашивал `ffprobe` о неизменившемся файле на каждое движение.

Отдельно про фейдер: во время воспроизведения он **стоит**, а по волне едет белая линия. Позиция считается арифметикой — точка старта плюс прошедшее время, — поэтому её можно спрашивать десять раз в секунду. Точка, уехавшая бы во время прослушивания, потом ставилась бы руками заново.

Про «держать FFmpeg в памяти». Запуск бинарника стоит **0,02 с** — держать резидентным нечего. Платится за открытие файла и декодирование, и вот их держать имеет смысл.

Держится **вся звуковая дорожка**, а не выбранные куски: заранее неизвестно, работают с началом, серединой или концом, и любая догадка будет неверной ровно тогда, когда понадобится. Материал раздела делает это дешёвым — запись Zoom весит копейки, двадцатиминутный проектный ролик с PCM даёт 55 МБ, полчаса — 82.

Длительность	Звук в памяти	Загрузка
20 минут	55 МБ	~1,3 с
30 минут	82 МБ	2,0 с
1 час	165 МБ	~3,9 с
4 часа	659 МБ	~15,7 с

Порог — **три часа**: выше него дорожка не держится целиком, и работают блоки по две минуты (5,5 МБ на блок, не больше шести сразу). Ими же отвечают первые секунды, пока полная дорожка ещё грузится в фоне, — фейдер её не ждёт никогда.

Моно 24 кГц выбрано намеренно: под курсором распознают речь, а не сводят фонограмму, и вчетверо меньше байтов приходят вчетверо раньше.

Как рисуется волна. Проверено сравнением на одних и тех же десяти секундах речи: `lin` — ниточка со всплесками, `sqrt` — тонкая форма, `log` — сплошная заливка без структуры, `cbrt` — тело, в котором читается фраза. Нормализация идёт по пику **всей дорожки**, а не окна: пик окна перерисовывал бы масштаб на каждом движении, и два места одной записи стало бы нельзя сравнить глазом. Пик, а не RMS — RMS рисует полнее, но заглаживает атаки, а рез целятся именно в них. Пик считается по дорожке в памяти: 86 мс на получасовом файле, дальше из кэша.

Полосу надо рисовать в том размере, в котором её показывают. Она рисовалась шириной 2560 px «с запасом под 4K», а на экране занимала 625: сжатие вчетверо безобидно для формы и смертельно для переходного процесса. Щелчок длиной 20 мс — это один пиксель в отрисовке и четверть пикселя на экране, и фильтр браузера его выбрасывает. На тест-файле с щелчком раз в секунду это выглядело так: в картинке тридцать щелчков, на экране около двадцати, а в десятисекундном окне из десяти оставалось **три**.

	В отрисовке	**Видно на экране**
Полоса «весь файл», 30 щелчков, рендер 2560 px	30	~20
Окно 10 секунд, 10 щелчков, рендер 2560 px	10	**3**
То же после правки (рендер по ширине показа)	30 / 10	**30 / 10**

Панель измеряет свою полосу и просит **ровно столько столбцов, сколько в ней CSS-пикселей**. Умножать на масштаб дисплея нельзя — я попробовал и получил ту же ошибку в меньшем масштабе: браузер раскладывает картинку в CSS-пикселях, и всё, что шире, он снова сжимает, снова теряя щелчки (из десяти доходило семь). Один столбец на один CSS-пиксель: ничего не пересэмплируется — ничего не теряется.

Десятисекундное окно убрано. Вторая полоса не успевала за первой: её перерисовка приходила позже, и две картинки расходились достаточно часто, чтобы вводить в заблуждение. Точность, ради которой она была, лучше даёт то, что и так есть — шаг по кадрам, клавиши и звук кадра под курсором. На освободившееся место переехал транспорт, который прежде стоял под всем блоком, вдали от полосы, которой управляет. На очень длинных роликах точку теперь ищут слухом и покадровым шагом.

Как играть куски, чтобы они играли. Первая схема отдавала каждый кусок элементу `<audio>` новым источником — то есть на каждое движение фейдера начиналась загрузка, а следующая её отменяла. `play()` отклонялся почти всегда: звук «прорывался раз в пять секунд», а реверс не работал вовсе. Ошибка архитектурная, а не в мелочах: проигрыватель нельзя собирать из подменяемых источников.

Так это делают монтажные программы, и так теперь здесь: дорожка **один раз** скачивается в браузер (HTTP-роут отдаёт тот же WAV, что лежит в памяти сервера) и декодируется в Web Audio; дальше каждый кусок — чтение из готового буфера. Ничего не грузится, отменять нечего.

	`<audio>` с подменой источника	Web Audio из буфера
20 кусков подряд по 40 мс	звучали единицы	**20 из 20**
Реверс	не работал	работает
Переключение файла	тишина до перезагрузки панели	дорожка подхватывается сама

Порог: дорожка длиннее 25 минут в браузер не отдаётся — декодированная, она стоит там четыре байта на сэмпл. Для таких файлов остаётся прежний путь с кусками через сокет.

Скраб назад звучит назад. Отматывание фейдера отдаёт тот же кадровый кусок, развёрнутый во времени, — так ведёт себя jog-колесо, и направление движения слышно, а не угадывается. Сэмплы уже в памяти, разворот стоит одного обращения к массиву.

Кадр превью — под курсором, а не на ключевом. Раньше картинка бралась на том ключевом кадре, куда ляжет рез: на all-intra это то же самое, а на long-GOP 4K ключевые идут раз в полторы секунды, и картинка отставала от руки и шла ступенями — искать по ней нельзя. Теперь кадр показывает точку, а ключевой кадр и расстояние до него живут в подписи: «где я» и «где начнётся кусок» — разные вопросы.

Кусок длиной в кадр, выровненный по кадру. Так делает монтажная программа: под курсором слышен именно тот кадр, на котором стоишь, а не секунда после него. На 23,976 это **41,7 мс**, на 25 — 40. Начало куска сдвигается вниз до границы кадра: полкадра расхождения на голове слова — это уже другой слог.

	Фиксированные 0,9 с	Кадровый кусок
Длительность на 23,976	900 мс	**41,7 мс**
Байтов на движение	57 КБ	**2 КБ**
Что слышно	секунда после точки	кадр под точкой

3.2. Час материала и отмена на живом FFmpeg

Всё предыдущее меряно на клипах до минуты. Часовой файл собран склейкой копий **без перекодирования**: 66 копий 55-секундного 4K дали **1 час 25 с, 50,7 ГБ**, H.264 3840×2160 на 23,976.

Что	На часовом файле
Прочитать факты о файле	0,43 с (повторно 0,28 с)
Ключевые кадры на 58-й минуте	0,47 с, повторно из кэша **0,00 с**
Кадр превью на 58-й минуте	1,11 с
Звуковая дорожка целиком в память	**166 МБ**, 5–16 с в фоне
Кусок звука в любой точке (0,5 / 30 / 58 мин)	**0,00 с**
Пик всей дорожки	9,77 с → **0,07 с** (см. ниже)
Волна окна на 58-й минуте	0,14 с
Рез 10 с на 58-й минуте	**254 кадра, +0,000 с**, 2,3 с работы
Рез последних 10 с файла	**358 кадров, +0,000 с**, 2,5 с работы

Пик считался девять секунд. Цикл на Python по 83 млн сэмплов — и он стоит перед первой отрисовкой волны. `audioop.max` делает тот же проход на C за **0,06 с**, ответ тот же. Модуль удаляют в Python 3.13, поэтому он используется при наличии, а иначе возвращается цикл, читающий каждый восьмой сэмпл: для масштаба отрисовки прореживание безобидно, для чего-то точного было бы нельзя.

Отмена посреди записи. Механизм был написан, но никогда не проверялся на живом FFmpeg. Проверка: рез получаса из часового файла, отмена через 6 секунд.

- дочерний процесс гасится корректно (`terminate`, пять секунд, затем `kill`);
- операция возвращается как несостоявшаяся, а не как успех;
- **но на диске оставалось 6 ГБ нечитаемого файла** — под именем готового куска. Мультиплексор не успел записать индекс, `moov atom not found`, ни один плеер такое не откроет, а в папке оно выглядит результатом.

Прежнее правило удаляло только файлы нулевой длины, рассуждая, что частичный результат может быть нужен. Это верно ровно до тех пор, пока частичный файл открывается. Теперь

проверяется читаемость: что открывается — остаётся (и в журнале сказано, до какой секунды), что не открывается — удаляется с объяснением. После правки отмена не оставляет ничего.

---

4. Синхрон звука: считается в миллисекундах

Меряется не «есть ли рассинхрон» вообще, а **изменил ли его рез**: у любого источника есть своя задержка, и абсолютное значение ничего не говорит.

Материал — клип, где метка есть в обеих дорожках одновременно: белый кадр в начале каждой секунды и щелчок 20 мс в тот же момент.

Контейнер	Изменение против исходника
MP4	0…2 мс
MOV	0…5 мс
MKV	1…3 мс
MXF	0…5 мс (после перезаписи PCM; до неё было 9…18 мс)

Про MXF отдельно. При простом копировании PCM контейнер приносит с собой сэмплы от границы чанка до точки реза — **384 сэмпла, 8 мс звука**, который относится к моменту **до** реза. Перезапись того же PCM той же разрядности кладёт звук на сэмпл и совпадает с эталонным резом байт в байт. Поколение не теряется: PCM переписывается в PCM.

Ещё одна мелочь того же порядка: MXF принимает PCM **только** little-endian. Canon пишет `pcm_s16be`, поэтому при выводе в MXF байты переставляются — те же сэмплы, порядок, которого требует контейнер.

---

5. Проверка «без потерь»

Заявление «поток скопирован, не пережат» проверяется, а не декларируется.

Картинка. Хэши декодированных кадров куска и того же диапазона источника (`-f framemd5`) — совпадение до байта.

Звук. Жёстче: вытащить звук куска и звук всего источника в raw PCM и поискать первый дословно во втором. На камерном файле нашёлся — и позиция вхождения дала точное

смещение реза: **48 сэмплов, 1 мс, сороковая доля кадра**.

Картинка на входе куска (open-GOP). Число кадров ничего не говорит о том, что видно. При open-GOP кадры после ключевого могут ссылаться на картинки до него, и кусок, начатый оттуда, показывает их битыми — обычная претензия к копированию long-GOP. Проверка прямая: сравнить хэши **декодированных** кадров куска с теми же кадрами источника, считая от ключевого кадра, на который лёг вход.

Источник	Вход куска	Сравнено	Совпало
HEVC long-GOP 29,97 (`has_b_frames=2`)	ключевой на 10,010 с	40 кадров	**40**
Камерный 4K H.264 23,976	ключевой на 18,852 с	40 кадров	**40**

Бит в бит, без единого расхождения: на входе куска видно ровно то же, что в источнике.

---

6. Контейнер и кодек: что во что кладётся

Каждая ячейка — запись одной секунды с `-c copy` и **чтение результата обратно** со сверкой кодека. Проверять по коду возврата нельзя: файл может записаться и не читаться.

Видео	MP4	MOV	MKV	MXF	TS	M2TS
H.264	да	да	да	да	да	да
HEVC	да	да	да	**нет**	да	да
ProRes	**нет**	да	да	да	**нет**	**нет**
DNxHD/DNxHR	**нет**	да	да	да	**нет**	**нет**
MPEG-2	да	да	да	да	да	да
AV1	да	**нет**	да	**нет**	**нет**	**нет**
VP9	да	**нет**	да	**нет**	**нет**	**нет**

Звук	MP4	MOV	MKV	MXF	TS	M2TS
PCM 16 LE	да	да	да	да	**нет**	**нет**

Звук	MP4	MOV	MKV	MXF	TS	M2TS
PCM 16 **BE** (`twos`, Canon)	да	да	да	**нет**	**нет**	**нет**
PCM 24 LE	да	да	да	да	**нет**	**нет**
PCM 32 float	да	да	да	**нет**	**нет**	**нет**
AAC	да	да	да	**нет**	да	да
AC-3 / E-AC-3	да	да	да	**нет**	да	да
MP3	да	да	да	**нет**	да	да
FLAC	да	**нет**	да	**нет**	**нет**	**нет**
ALAC	да	да	да	**нет**	**нет**	**нет**
Opus	да	**нет**	да	**нет**	да	да

Таблицы пересняты на 9.0.1 целиком и совпали с 8.0.1 до ячейки.

---

7. Метаданные: что переживает рез

Что	MP4	MOV	MKV
Поворот (`rotation=90`)	да	да	да
Язык дорожек	да	да	да
Теги проекта (title, artist)	да	да	да
Камерные теги (make, model, brands)	да	да	частично
Таймкод дорожкой `tmcd`	да	да	**нет** (текстовый тег)

`-map_metadata 0` **не переносит** камерные теги в MP4 — ни `make`, ни `model`, ни бренд вроде `mp42hvc1CAEP`: мультиплексор пишет свои `isomiso2mp41`. Нужен `-movflags use_metadata_tags`.

Таймкод при копировании MOV → MP4 переносится дорожкой, но **значение остаётся прежним** — сдвиг на длину отрезанного надо писать самому.

8. Предел инструмента: камерный RAW

Два формата, которые FFmpeg не открывает ни в какой версии:

Файл	Внутри	Что видит ffprobe
Sony BURANO X-OCN LT	X-OCN в MXF OP1a	поток без имени кодека, `width=0`; звук `pcm_s24le` читается
ARRI Alexa Mini `.mxf`	ARRIRAW в MXF	то же самое

Простое копирование даёт «dimensions not set» и файл нулевой длины. Проверено на 8.0.1 и на 9.0.1 — одинаково. Такой материал режется в родном ПО производителя или после транскода, и честнее сказать это до запуска, чем показать нулевой файл.

Тот же ARRI в **MOV** (ProRes 4444 XQ, 2944×2160, 12 бит, 5 дорожек PCM) режется кадр в кадр вместе с таймкодом.

Отдельная мелочь оттуда же: ARRI пишет цвет как GBR с зарезервированными primaries, и `swscale` отказывается делать из этого миниатюру — «Unsupported input». Кадр для превью приходится просить, назвав цветовое пространство явно.

9. Растяжение времени: фильтр съедает 15 мс

Проверялось для пересчёта частоты, где звук тянется вслед за картинкой.

Отношение	Источник 10 с	Источник 0,25 с
60 → 24	−0,10 %	−3,28 %
120 → 24	−0,13 %	−5,11 %
1000 → 24	−0,16 %	−6,04 %
8000 → 24	−0,16 %	−6,14 %

Потеря зависит **не от коэффициента, а от длины входа**: и `atempo`, и `rubberband` съедают около **15 мс** исходного звука — это их внутренний буфер. На реальном материале это доли процента; на четвертьсекундном обрывке при растяжении в сотни раз — уже секунды.

Дописывание тишины на вход (`apad`) возвращает часть, но вносит собственный перелёт, поэтому фильтр оставлен как есть, а граница записана здесь.

Заодно проверено, что источник может быть любым: 8000 кадров в секунду → 24 прошли без потери единого кадра (2000 кадров, ровно 83,3333 с).

---

10. Грабли FFmpeg, найденные измерением

Ни одна не очевидна из документации, и каждая стоила отдельного разбора.

№	Грабля	Проявление
1	`-ss` до `-i` сбрасывает время	`-to` после него считает от нового нуля
2	`round(fps)` на дробных частотах	4 минуты на 29,97 превращаются в 240,27 с
3	Округление конца «к ближайшему»	теряется кадр, стоящий на 29,9966
4	`-t` при копировании	перебор на 1–3 кадра на потоке с B-кадрами
5	`-frames:v` точен только с точным `-ss`	при seek мимо ключевого кадра кусок короче
6	HEVC с B-пирамидой	муксер пишет на 2–4 кадра меньше запрошенного
7	`-avoid_negative_ts make_zero`	видео уезжает на 0,1 с от звука
8	`start_time` ≠ 0 (MPEG-TS 1,44 с)	все точки смещаются на эту величину
9	`avg_frame_rate` на VFR	12 запрошенных секунд превращаются в 16
10	`-n` на существующий файл	«already exists» и **код возврата 0**
11	PCM при копировании в MXF	384 лишних сэмпла (8 мс) в начале
12	`-map_metadata 0` в MP4	камерные теги и бренды теряются
13	`pan=mono|c0=c1` на моно	пишет тишину, код возврата 0
14	MXF + сжатый звук	«Could not write header», код 4294967274
15	Пакет ≠ кадр	600 записано, 597 декодируется
16	Камерный RAW в MXF	«dimensions not set», файл нулевой длины

№	Грабля	Проявление
17	Предупреждение в `stderr` ломает разбор JSON	`json.loads` падает на файле, который читается прекрасно

---

11. Случаи, когда измерение соврало

Здесь стоит быть особенно честным: каждый из этих выводов выглядел убедительно, и каждый оказался неверным.

«MKV уводит звук на 107 мс». Сравнивались «первая найденная вспышка» и «первый найденный щелчок». Один поток открывался меткой, другой нет — детекторы описывали **разные события**. Оба потока были сдвинуты вместе, рассинхрона не было. Правильно — сопоставлять каждую вспышку со своим щелчком и смотреть на весь набор. Предупреждение, добавленное в продукт по этому «открытию», пришлось снимать.

«ProRes нельзя положить в MXF». Проверка шла с партнёрским звуком AAC, который MXF не принимает вовсе. Ячейка показывала отказ партнёра, а не проверяемого кодека. Партнёрский поток надо подбирать под контейнер.

«HEVC режется точно». Считались пакеты. Кадров на три меньше — см. раздел 3.

«Звук скопирован дословно» — дважды подряд ни о чём. Побитовость проверяется поиском сэмплов куска внутри источника, и оба раза поиск находил совпадение, которое ничего не доказывало. Сначала сигналом была синусоида: она периодична, любой её кусок совпадает с началом, и «найдено на 0,000 с» получается всегда. Потом клип имел единственный ключевой кадр — в нуле, так что рез «с третьей секунды» физически начинался с начала файла, и совпадение снова выходило честным по форме и пустым по смыслу. Проверять надо на неповторяющемся сигнале (розовый шум) и на клипе с частыми ключевыми кадрами (`-g 25`): тогда совпадение находится ровно на запрошенной секунде, и это уже что-то значит.

Отдельно: `blackdetect` работает целыми кадрами, `silencedetect` — окнами; вместе они дают около 40 мс погрешности, то есть **больше**, чем величина, которую обычно меряют. На них легко увидеть сдвиг, которого нет.

---

12. Чем это меряется

```
# кадры: пакеты (быстро) и декодированные (честно)
ffprobe -v error -select_streams v:0 -count_packets -show_entries
stream=nb_read_packets -of csv=p=0 FILE
ffprobe -v error -select_streams v:0 -count_frames  -show_entries
stream=nb_read_frames  -of csv=p=0 FILE

# ключевые кадры вокруг точки, без скана всего файла
ffprobe -v error -select_streams v:0 -skip_frame nokey -read_intervals
"START%+WINDOW" -show_entries frame=pts_time -of csv=p=0 FILE

# положение вспышки: обязательно с passthrough, иначе фильтр дублирует кадры под свои
часы
ffmpeg -v info -i FILE -map 0:v:0 -vf
"signalstats,metadata=print:key=lavfi.signalstats.YAVG" -fps_mode passthrough -f null
-

# звук по сэмплам, а не детектором: точность 20 мкс
ffmpeg -v error -i FILE -map 0:a:0 -ac 1 -ar 48000 -f s16le -

# побитовая проверка копирования
ffmpeg -v error -i OUT -map 0:v:0 -frames:v 100 -f framemd5 -
```

Пробы читать **с разделёнными потоками**: Sony MXF встречает каждое чтение строкой «could not resolve file descriptor strong ref», и если `stderr` подмешан в `stdout`, разбор JSON падает на файле, который читается прекрасно.

4.1. Дрейф: его нет, и вот чем это показано

Синхрон в разделе 4 меряли на клипах. Вопрос был поставлен иначе: **уезжает ли звук от картинки по мере длины файла** — то, что в монтаже всплывает через полчаса и портит всё.

Час материала (120 копий тридцатисекундного дубля, склеенных без перекодирования, 696 МБ), и из него вырезано по 10 секунд в трёх местах:

Откуда режем	Картинка	Звук	Разница
10-я секунда	10,000000	9,990000	−10 мс

Откуда режем	Картинка	Звук	Разница
30-я минута	10,000000	9,984000	−16 мс
59-я минута	10,000000	10,000000	**0 мс**

Величина не растёт. В конце часа она ровно ноль, в середине — шестнадцать миллисекунд; это не накопление, а то, где пришлась граница аудиопакета (21 мс на этом файле). Дрейф выглядел бы иначе: 10 мс в начале, 300 в конце.

Начало обоих потоков во всех трёх кусках — **0,000000**: рез не сдвигает звук относительно картинки, он их только вместе укорачивает.

Так и должно быть, и причина простая: потоки **копируются**. Пакеты переносятся со своими исходными метками времени, часы остаются часами источника, и умножать их не на что. Дрейф появляется там, где время пересчитывают — при перекодировании или растяжении, — и там он у нас измерен отдельно (раздел 9: фильтры съедают 15 мс независимо от коэффициента).

Недобор в конце куска убирается там, где он мешает: при перезаписи PCM звук режется по сэмплу, и на вещательном MXF картинка и звук вышли одинаковыми до микросекунды — 3,003000 и 3,003000 (раздел 12.1).

Ловушка этого измерения

Первый замер сравнивал последний **пакет** звука с последним пакетом картинки и дал бессмыслицу: «звук на 40 мс длиннее». У видео с B-кадрами порядок хранения не совпадает с порядком показа — последние три пакета шли 9,84 / 9,96 / 9,92, — и «последний пакет» оказался не последним кадром. Сравнивать надо длительности потоков, а не хвосты пакетов.

12.1. Вещательный MXF: 4K, десять потоков, служебные данные

Два клипа с реальной съёмки, `A004C012_201031NW.mxf` и `A006C010_201031JD.mxf` (98 и 281 МБ на 3,4 и 6,2 секунды — 243 и 378 Мбит/с). Материал плотнее всего, что мерили до сих пор:

Что	Значение
Видео	H.264 **all-intra**, 4096×2160, `yuv422p10le`, 23,976
Звук	**восемь** отдельных моно-дорожек PCM 24 бита / 48 кГц

Что	Значение
Данные	`smpte_436m_anc` — служебные данные SDI
Таймкод	06:49:06:03 и 04:13:14:11 — настоящие съёмочные

Каждый кадр — ключевой. Прилипания начала здесь не бывает вовсе: запрос на 00:00:01,000 лёг на кадр 00:00:01,001, то есть на ближайший кадр, а не на далёкий опорный.

Рез `MXF → MXF`, оба файла:

Файл	Запрос	Кадров	Расхождение	Таймкод
A006C010	1,000 → 4,000	72	**+0,000 с**	06:49:06:03 → 06:49:07:03
A004C012	0,500 → 2,500	48	**+0,000 с**	04:13:14:11 → 04:13:14:23

В результате **все десять потоков**: картинка, восемь моно-дорожек и служебные данные. Хэш первого кадра результата совпал с хэшем кадра источника в точке реза — `1105dab6b3f5c81761e95784e58a1bd8` в обоих. Звук переписан по сэмплу, как для MXF и положено.

Что здесь выяснилось про поток данных

Правило «какие контейнеры держат поток данных» было написано по предположению и оказалось неверным дважды:

1. **MXF держит.** Считалось, что нет. Вещательный оригинал несёт дорожку SMPTE 436M и принимает её обратно без единого возражения.
2. **Контейнера мало — важно происхождение.** MOV держит потоки данных вообще, но эту дорожку не принимает: рез `MXF → MOV` **упал** с кодом 4294967274 и без внятного сообщения. Поток данных пишется под тот формат, в котором живёт, и переезжает только в такой же.

Теперь при смене контейнера раздел говорит об этом **до** запуска и режет дальше, а не падает:

```
[WARNING] The data stream (smpte_436m_anc) was written for MXF and MOV will not
          take it, so it stays behind. Keeping the container as MXF carries it
          over.
```

Отдельная мелочь, стоившая невнятного сообщения: у MXF-потока пустой четырёхсимвольный тег, и `ffprobe` отдаёт его как `[0][0][0][0]`. В сообщении теперь стоит имя кодека —

`smpte_436m_anc`, — а тег используется там, где он осмысленный: `gpmd`, `tmcd`.

---

13. Чего мы не проверяли

Список открытый — чтобы никто, включая автора, не решил, что тема закрыта:

- камеры: **вещательный MXF 4K all-intra с восемью дорожками и служебными данными проверен** (раздел 12.1), Canon HEVC и ARRI ProRes проверены раньше; остаются Panasonic, BMPCC, GoPro с телеметрией, AVCHD прямо с карты;
- файлы **длиннее** часа (час проверен, см. раздел 3.2);
- нехватка места на диске в середине записи;
- источник на сетевом пути или занятый другой программой;
- две операции одновременно;
- звук 96 кГц (в видео редкость; 32 бита float на 48 кГц — обычный случай с рекордеров — проверен: `pcm_f32le` переживает рез, звук куска найден в источнике дословно на 3,0000 с, смещение **+0,00 мс**);
- звук, уходящий за последний кадр на границу пакета (0,1 с после добора хвоста): об этом сообщается, но на слух не проверялось.

Устройство раздела «Обрезка» и его отказы — в TRIM_MATRIX_RU.md, замысел и решения — в PLAN_TRIM_RU.md и DECISIONS.md. Методика измерений и разбор Shutter Encoder — в `E:\TOOLS\fps-test`.