Swiss Army knife

From Wikipedia, the free encyclopedia
(Redirected from Swiss army knife)

The **Swiss Army knife** (French: *couteau suisse*, German: *Schweizer Offiziersmesser*: "Swiss officer's knife", Swiss-German: *Sackmesser*, Italian: *Coltellino svizzero*) is a brand of pocket knife or multi-tool manufactured by Victorinox AG (and Wenger SA). The term "Swiss Army knife" was coined by US soldiers after World War II due to the difficulty they had in pronouncing the German name.[1]

The Swiss Army knife generally has a sharp blade, as well as various tools, such as screwdrivers, a can opener, and many others. These attachments are stowed inside the handle of the knife through a pivot point mechanism. The handle is usually red, and features a Victorinox or Wenger "cross" logo or, for Swiss military issue knives, the coat of arms of Switzerland.

Originating in Ibach, Switzerland, the Swiss Army knife was first produced in 1891 after the company, Karl Elsener, which later became Victorinox, won the contract to produce the Swiss Army's *Modell 1890* knife from the previous German manufacturer. In 1893, the Swiss cutlery company, Paul Boéchat & Cie, which later became Wenger, received its first contract from the Swiss military to produce model 1890 knives; the two companies split the contract for provision of the knives from 1908 until Victorinox acquired Wenger in 2005.

The design of the knife and its versatility have both led to worldwide recognition.[2]

Victorinox "Huntsman" Swiss Army knife, with knife chain and belt clip.

Wenger RangerGrip 75 Swiss army knife.

Contents

History [edit]

Origins [edit]

During the late 1880s, the Swiss Army decided to purchase a new folding pocket knife for their soldiers. This knife was to be suitable for use by the army in opening canned food and disassembling the Swiss service rifle, the Schmidt-Rubin M1889, which required a screwdriver for assembly.

Modell 1890, the first Swiss Soldier Knife produced by Wester & Co. Solingen

In January 1891, the knife received the official designation *Modell 1890*. The knife had a blade, reamer, can-opener, screwdriver, and grips made out of dark oak wood that was later partly replaced with ebony wood. At that time no Swiss company had the necessary production capacity, so the initial order for 15,000 knives was placed with the German knife manufacturer Wester & Co. from Solingen, Germany. These knives were delivered in October 1891.

In 1891, Karl Elsener, then owner of a company that made surgical equipment, set out to manufacture the knives in Switzerland itself. At the end of 1891 Elsener took over production of the *Modell 1890* knives, but Elsener was not satisfied

中文
Edit links

with its first incarnation. In 1896, Elsener succeeded in attaching tools on both sides of the handle using a special spring mechanism: this allowed him to use the same spring to hold them in place, an innovation at the time.[3] This allowed Elsener to put twice as many features on the knife. On 12 June 1897 this knife featuring a second smaller cutting blade, corkscrew, and wood fiber grips was originally registered with the patent office as The Officer's and Sports Knife, though it was never part of a military contract.[4]

Karl Elsener used the cross and shield to identify his knives, the symbol still used today on Victorinox-branded versions. When his mother died in 1909, Elsener decided to name his company "Victoria" in her memory. In 1921 the company started using stainless steel to make the Swiss Army Knife. Stainless steel is also known as "inox", short for the French term *acier inoxydable*.[5] "Victoria" and "inox" were then combined to create the company name "Victorinox".[6] Victorinox's headquarters and show room are located in the Swiss town of Ibach.

Victorinox and Wenger [edit]

Elsener, through his company Victorinox, managed to control the market until 1893, when the second industrial cutler of Switzerland, *Paul Boéchat & Cie*, headquartered in Delémont in the French-speaking region of Jura, started selling a similar product. This company was later acquired by its then General Manager, Theodore Wenger, and renamed the Wenger Company. In 1908 the Swiss government, wanting to prevent an issue over regional favouritism, but perhaps wanting a bit of competition in hopes of lowering prices, split the contract with Victorinox and Wenger, each getting half of the orders placed. By mutual agreement, Wenger has advertised as the *Genuine Swiss Army Knife* and Victorinox used the slogan, the *Original Swiss Army Knife*.

On April 26, 2005, Victorinox acquired Wenger, becoming once again the sole supplier of knives to the Swiss Armed Forces. Victorinox had kept both consumer brands intact, but on January 30, 2013, Wenger and Victorinox announced that the separate knife brands were going to be merged into one brand: Victorinox. Wenger's watch and licensing business will continue as a separate brand.[7]

After World War II [edit]

According to Carl Elsener, head of Victorinox in 2009, US soldiers bought Swiss Army knives in huge numbers at PX stores on military bases following the conclusion of the Second World War. As German: *Schweizer Offiziersmesser* was too difficult for them to say, they called it the "Swiss Army knife", the name by which the knife is now most commonly known worldwide.[2]

Swiss military knife contract [edit]

In 2007, the Swiss Government made a request for new updated soldier knives for the Swiss military for distribution in late 2008. The evaluation phase of the new soldier knife began in February 2008, when Armasuisse issued an invitation to tender. A total of seven suppliers from Switzerland and other countries were invited to participate in the evaluation process. Functional models submitted by suppliers underwent practical testing by military personnel in July 2008, while laboratory tests were used to assess compliance with technical requirements. A cost-benefit analysis was conducted and the model with the best price/performance ratio was awarded the contract. The order for 75,000 soldier knives plus cases was worth SFr 1.38 million. This equates to a purchase price of SFr 18.40, €12.12, GB£17.99 in October 2009 per knife plus case.


The one-hand knife of the German Army as issued by the *Bundeswehr* since 2003.

Victorinox won the contest with a knife based on the One-Hand Germany Army Knife as issued by the German *Bundeswehr* and released in the civilian model lineup with the addition of a toothpick and tweezers stored in the nylon grip scales (side cover plates) as the One-Hand Trekker/Trailmaster model. Mass production of the new *Soldatenmesser 08* (Soldier Knife 08) for the Swiss Armed Forces was started in December 2008.[8]

Features [edit]

Tools [edit]

Various models of Swiss Army knives exist, with different tool combinations for specific tasks designed for everyday carry. The simplest model sold includes only a single blade. The most common tools featured are, in addition to the main blade, a smaller second blade, tweezers, toothpick, corkscrew, can opener, bottle opener, slotted/flat-head screwdriver(s), phillips-head screwdriver, nail file, scissors, saw (regular, wood), file, reamer, hook (parcel carrier, tightening aid for shoelaces, etc.), magnifying glass, ballpoint pen, fish scaler, hex wrench (with drive bits), pliers, gimlet, compass, ruler and keyring.


The Victorinox Cybertool

Recent technological features include USB flash drives, digital clock, digital altimeter, LED light, laser pointer, and MP3 player.[*citation needed*] The Victorinox Cybertools series includes features for use with computers and electronic equipment. In addition to the usual tools, including pliers and scissors, they have a 4 mm hex screwdriver bit holder and bit case with 4 double-ended bits (8 ends).[9]

In January 2010 Victorinox announced the Presentation Master model line to be released in April 2010. The technologically most advanced model includes a laser pointer, a 32 GB detachable flash drive and Bluetooth.[*further explanation needed*] One of the Presentation Master models will be the bladeless Presentation Master Flight model. Besides being lightweight, protected and portable, this bladeless

Presentation Master version is permitted to be carried on airplanes. For added convenience, the flash drive component is removable, offering travelers the option to carry their data while storing their pocket tool in their checked baggage.[10]


The Giant by Wenger

Since 2006 Wenger has produced a knife called "The Giant" that includes every implement the company has ever made.[11] With 87 tools and 141 different functions it is recognised by the Guinness Book of Records as the world's most multifunctional penknife.[12] It retails for about EUR 798 or USD 1,000, though some vendors demand much higher prices.[13]

In the same year Victorinox released SwissChamp XAVT which includes 80 functions, with retail price of USD 425.[14] The Guinness Book of Records recognizes a unique 314-blade Swiss Army-style knife made in 1991 by Master Cutler Hans Meister as the world's largest penknife, weighing 11 pounds.[2]

Locking mechanisms [edit]

Some Swiss Army knives feature a locking mechanism for one or two tools. The employed locking systems make an accidental closure during use of an extended tool unlikely. Several Wenger and Victorinox models feature a locking cutting blade secured by a slide lock that is operated with an unlocking-button integrated in the scales. Furthermore several models from the Victorinox 111 mm series feature a more robust double liner lock that secures the cutting blade and bottle opener.[15]


Dual liner lock system as used in the *Soldatenmesser 08* and various other Victorinox 111 mm models

Design and materials [edit]

Rivets and flanged bushings made from brass hold all machined steel parts and other tools, separators and the scales together. The rivets are made by cutting and pointing appropriately sized bars of solid brass.

The separators between the tools have been made from aluminium alloy since 1951. This makes the knives lighter. Previously these separating layers were made of nickel-silver.[16]

The martensitic stainless steel alloy used for the cutting blades is optimized for high toughness and corrosion resistance and has a composition of 15% chromium, 0.60% silicon, 0.52% carbon, 0.50% molybdenum, and 0.45% manganese and is designated X55CrMo14 or DIN 1.4110 according to Victorinox.[17] After a hardening process at 1040 °C and annealing at 160 °C the blades achieve an average blade steel hardness of 56 HRC. This steel hardness is suitable for practical use and easy resharpening, but less than achieved in stainless steel alloys used for blades optimized for high wear resistance. According to Victorinox the martensitic stainless steel alloy used for the parts is X39Cr13 or DIN 1.4031 and for the springs X20Cr13 or DIN 1.4021.[18][19]

The steel used for the wood saws, scissors and nail files has a steel hardness of HRC 53, the screwdrivers, tin openers and awls have a hardness of HRC 52, and the corkscrew and springs have a hardness of HRC 49.[18][*citation needed*]

The metal saws and files, in addition to the special case hardening, are also subjected to a hard chromium plating process so that iron and steel can also be filed and cut.[16][20]

Although red Cellulose Acetate Butyrate (CAB) (generally known trade names are Cellidor, Tenite and Tenex) scaled Swiss Army Knives are most common, there are many colors and alternative materials like nylon and aluminium for the scales available.[21][22] Many textures, colors and shapes now appear in the Swiss Army Knife. Since 2006 the scales on some knife models can have texturized rubber non-slip inlays incorporated, intended for sufficient grip with moist or wet hands. A modding community has also developed, resulting in custom models produced with colorful anodized patterns or wood handles.[*citation needed*]


Various sized and scaled models produced by Wenger

Assembly [edit]

During assembly, all components are placed on several brass rivets. The first components are generally an aluminum separator and a flat steel spring. Once a layer of tools is installed, another separator and spring are placed for the next layer of tools. This process is repeated until all the desired tool layers and the finishing separator are installed. Once the knife is built, the metal parts are fastened by adding brass flanged bushings to the rivets. The excess length of the rivets is then cut off to make them flush with the bushings. Finally the remaining length of the rivets is flattened into the flanged bushings.

After the assembly of the metal parts, the blades are sharpened to a 15° angle, resulting in a 30° V-shaped steel cutting edge. The blades are then checked with a laser reflecting goniometer to verify the angle of the cutting edges. Finally the scales are pressed onto the flanged bushings. The scales are being held in place by holes incorporated in the insides of the scales that result in a tight shape connection with the flanged bushings.[23]

Sizes [edit]

Victorinox Swiss Army knives use 58 mm (2.3 in), 74 mm (2.9 in), 84 mm (3.3 in), 91 mm (3.6 in), 93 mm (3.7 in), 100 mm (3.9 in), 108 mm (4.3 in) and 111 mm (4.4 in) closed length steppings. The thickness of the knives varies depending on the number of tool layers included. The 91 mm (3.6 in) knives offer the most variety in tool configurations in the Victorinox model line.[24]


Wenger EvoGrip S17 85 mm knife.

Wenger Swiss Army knives use 65 mm (2.6 in), 75 mm (3.0 in), 85 mm (3.3 in) 93 mm (3.7 in), 100 mm (3.9 in), 120 mm (4.7 in) and 130 mm (5.1 in) closed length steppings. Thickness varies depending on the number of tool layers included. The 85 mm (3.3 in) knives offer the most variety in tool configurations in

the Wenger model line.[25]

Manufacturers [edit]

Main articles: Victorinox and Wenger

The Swiss company Victorinox AG and up to 2008 its wholly owned subsidiary Wenger SA supply about 50,000 knives to the Military of Switzerland each year, and manufacture many more for export, mostly to the United States. Many commercial Victorinox and Wenger Swiss Army knives can be immediately distinguished by the "cross logos" depicted on their grip shells; the Victorinox cross is surrounded by a shield with bilateral symmetry, while the Wenger cross is surrounded by a slightly rounded square with quadrilateral symmetry. Since 1961 the military issue knives supplied to the Swiss Armed Forces bear the Swiss Coat of Arms.

Many other companies manufacture similar-looking multi-tool folding knives in a wide range of quality and prices. The cross-and-shield emblem and the words SWISS ARMY are registered trademarks of Victorinox AG and its related companies.[*citation needed*]

Knives issued by the Swiss military [edit]

Since the first issue as personal equipment in 1891 the *Soldatenmesser* (Soldier Knives) issued by the Swiss Armed Forces have been revised several times. There are five different main *Modelle* (models). Their model numbers refer to the year of introduction in the military supply chain. Several main models have been revised over time and therefore exist in different *Ausführungen* (executions), also denoted by the year of introduction. The issued models of the Swiss Armed Forces are:[26]

- Modell 1890
  - Modell 1890 Ausführung 1901
- Modell 1908
- Modell 1951
  - Modell 1951 Ausführung 1954
  - Modell 1951 Ausführung 1957
- Modell 1961
  - Modell 1961 Ausführung 1965
  - Modell 1961 Ausführung 1978
  - Modell 1961 Ausführung 1994
- Soldatenmesser 08 (Soldier Knife 08)

Soldier Knives are issued to every recruit or member of the Swiss Armed Forces and the knives issued to officers have never differed from those issued to non-commissioned officers or privates.[27] A model incorporating a corkscrew and scissors was produced as an officer's tool, but was deemed not "essential for survival", leaving officers to purchase it individually.[2]

Soldier knife model 1890 [edit]

The Soldier Knife model 1890 had a spear point blade, reamer, can-opener, screwdriver and grips made out of oak wood scales (handles) that were treated with rapeseed oil for greater toughness and water-repellency, which made them black in color. The wooden grips of the Modell 1890 tended to crack and chip so in 1901 these were changed to a hard reddish-brown fiber similar in appearance to wood. The knife was 100 mm (3.9 in) long, 20.5 mm (0.81 in) mm thick and weighed 144 g (5.1 oz).[28]

Soldier knife model 1908 [edit]

The Soldier Knife model 1908 had a clip-point blade rather than the 1890s spear point blade, still with the fiber scales, carbon steel tools, nickle-silver bolster, liners, and divider. The knife was 100 mm (3.9 in) long, 16.5 mm (0.65 in) mm thick and weighed 125 g (4.4 oz). The contract with the Swiss Army split production equally between the Victorinox and Wenger companies.[28]

Modell 1908, the knife that was issued from 1908 to 1951.[28]

Soldier knife model 1951 [edit]

The soldier Knife model 1951 had fiber scales, nickle-silver bolsters, liners, and divider, and a spear point blade. This was the first Swiss Armed Forces issue model where the tools were made of stainless steel. The screwdriver now had a scraper arc on one edge. The knife was 93 mm (3.7 in) long, 13.5 mm (0.53 in) mm thick and weighed 90 g (3.2 oz).

Soldier knife model 1961 [edit]

The Soldier Knife model 1961 has a 93 mm (3.7 in) long knurled alox handle with the Swiss crest, a drop point blade, a reamer, a blade combining bottle opener, screwdriver, and wire stripper, and a combined can-opener and small screwdriver and it weighs 72 g (2.5 oz). The knife was 12 mm (0.47 in) mm thick and weighed 72 g (2.5 oz)

This official Swiss military model also contains a brass spacer, which allows the knife, with the screwdriver and the reamer extended simultaneously, to be used to assemble the SIG 550 and SIG 510 assault rifles: the knife serves as a restraint to the firing pin during assembly of the lock. The Soldier Knife model 1961 was manufactured only by Victorinox and Wenger.

Modell 1961, the knife that was issued from 1961 to 2008.

Soldier knife 08 [edit]

The Soldier Knife 08 was first issued to the Swiss Armed Forces beginning with the first basic training sessions of 2009.[29]

The Soldier Knife 08 features an 111 mm (4.4 in) long ergonomic handle with polymer texturized non-slip inlays incorporated in the nylon grip shells and a double liner locking system, one-hand 86 mm (3.4 in) long locking partly serrated chisel ground drop point blade, wood saw, can opener with small 3 mm (0.12 in) screwdriver, locking bottle opener with large 7 mm (0.28 in) screwdriver and wire stripper/bender, reamer, Phillips (PH2) screwdriver and keyring. The Soldier Knife 08 width is 34.5 mm (1.36 in), thickness is 18 mm (0.71 in), overall length opened is 197 mm (7.8 in) and it weighs 131 g (4.6 oz). The Soldier Knife 08 is manufactured only by Victorinox.

Soldatenmesser 08, the knife currently issued to the Swiss Armed Forces.

Knives issued by other militaries [edit]

The armed forces of more than 20 different nations have issued their own versions of the Swiss army knife, among them the forces of Germany, France, the Netherlands and Malaysia.[30]

Old German Army Knife

New German Army Knife

Malaysian Army Knife

Cultural impact [edit]

The Swiss Army knife has been added to the collection of the New York Museum of Modern Art (MoMA) and Munich's State Museum of Applied Art for its design. The term “Swiss Army” currently is a registered trademark owned by Victorinox AG and its subsidiary, Wenger SA.[31]

The television show MacGyver features Angus MacGyver who frequently uses a Swiss Army knife to solve problems and construct simple objects.

The term "Swiss Army knife" has entered popular culture as a metaphor for usefulness and adaptability.[32] The multi-purpose nature of the tool has also inspired a number of other gadgets.[33]

See also [edit]

- Baselard
- Ballpoint pen knife
- Multi-tool
- Gerber multitool
- Leatherman

Notes and references [edit]

1. ^ "Victorinox Swiss Army Knives Info" . Victorinox. Retrieved 2008-09-18.
2. ^ a b c d Foulkes, Imogen (2009-07-30). "From humble tool to global icon" . BBC News. Retrieved 2011-10-31.
3. ^ "Newtons Newton's Newton Swiss Switzerland Vaud French France Beaujolais Blace Gonnu 69460" . Newtons.ch. Retrieved 2011-11-01.
4. ^ SAKWiki Officer's and Sports Knife
5. ^ *Dictionnaire Des Sciences Et Techniques Du Pétrole: Anglais-français* . 1993. p. 427. ISBN 2710806487.
6. ^ "Victorinox official company history" . Retrieved 2007-12-10.
7. ^ Victorinox joins forces and integrates Wenger knife business
8. ^ Victorinox wins contract for new army knife
9. ^ "Home" . Victorinox AG. Retrieved 2011-10-27.
10. ^ "Victorinox USB Product Page Master/Secure" . Secure.victorinox.com. Retrieved 2011-10-31.
11. ^ "Giant Knife Wenger Swiss Army Knife" . Wenger.ch. Retrieved 2011-11-01.
12. ^ "Now That's a Knife: Swiss Army Knife Sets Record for Tools" . *Fox News*. 2007-11-23.
13. ^ Dual Time Zone Watches. "The Giant Guinness World Record Holding Swiss Army Knife by Wenger at Swiss Knife Shop" . Swissknifeshop.com. Retrieved 2011-10-27.
14. ^ "MultiTools — SwissChamp XAVT" . Victorinox Swiss Army. Retrieved 2011-10-31.
15. ^ SAKWiki. com Locking Systems
16. ^ a b "Victorinox Swiss Army Knives Info Steelinfo" . Pizzini.at. Retrieved 2011-10-31.
17. ^ Product presentation of a knife at www.victorinox.com
18. ^ a b e-mail scan regarding applied steel alloys by Robert Elsener from Victorinox
19. ^ "Stainless Steel: Tables of Technical Properties Second Edition 2007" (PDF). Euro Inox. 2007. Retrieved 2011-10-31.
20. ^ "Swiss Army Knife FAQ's at the Secret Order of Swiss Army Knives website" . Retrieved 2011-10-31.[*dead link*]
21. ^ US Patent 20130040125 A1 Expandable polymers of cellulose acetate butyrate
22. ^ "Cellídor resins" . Albis.com. Retrieved 2011-10-31.
23. ^ [1] [*dead link*]
24. ^ SAKWiki.com Victorinox Knife List

25. ^ SAKWiki.com Wenger Knife List
26. ^ "Schweizer Soldatenmesser 1890 - 2007 exhibition Schloss Thun — Schweiz (German)" . Messerforum.net. Retrieved 2011-10-31.
27. ^ McPhee, John (1983-10-31). "La Place de la Concorde Suisse-I" . *The New Yorker*. p. 50. Retrieved 22 July 2013.
28. ^ [a] [b] [c] The Swiss Army Knife Owner's Manual, published 7 September 2011
29. ^ "Neues Soldatenmesser 08 an die Armee übergeben", March 10, 2009
30. ^ Subramanian, Samanth (February 16, 2010): The Swiss Army Knife returns to the battlefield . Retrieved July 27, 2012.
31. ^ "Legal - Victorinox Swiss Army" . Retrieved 16 September 2012.
32. ^ "Victorinox AG — Company History" . Fundinguniverse.com. Retrieved 2011-10-31.
33. ^ "Swiss Army Knife Launches the Age of the Multitool" . Wired.com. 2008-06-12. Retrieved 2011-10-31.

External links [edit]

- Victorinox manufacturer's website
- Wenger manufacturer's website

Wikimedia Commons has media related to *Swiss Army knives*.

v · t · e ·	**Knives and daggers**
	List of daggers · List of blade materials ·
Types of knives	Aircrew Survival Egress Knife · Athame · Balisong/Butterfly · Ballistic · Ballpoint pen knife · Bayonet · Boline · Bolo · Boning · Boot knife · Bowie · Bread knife · Cane knife · Cheese knife · Chef's knife · Cleaver · Clip point · Combat knife · Commander · Corvo · CQC-6 · Dagger · Deba bōchō · Diving knife · Drop point · Electric knife · Fairbairn–Sykes fighting knife · Gerber Mark II · Ginsu · Grapefruit knife · Gravity knife · Gunong · Hunting dagger · Hunting knife · Jacob's ladder · Karambit · Kila · Kirpan · Kitchen knife · Kukri · Laguiole knife · Machete · Mandau · Mezzaluna · Misericorde · Mora knife · Multi-tool · Nakiri bōchō · Navaja · Neck knife · Opinel knife · Palette knife · Pantographic knife · Parang · Penknife · Penny knife · Pocket knife · Putty knife · Puukko · Rampuri · Rondel dagger · Sabatier · Sami knife · Santoku · SARK · Scalpel · Seax · Sgian dubh · Sharpfinger · Shiv · Sliding knife · Smatchet · SOG Knife · Straight razor · Strider SMF · Survival knife · **Swiss Army knife** · Switchblade · Taping knife · Throwing knife · Tomato knife · Trench knife · Tumi · Ulu · Utility knife · Warrior knife · X-Acto · Yanagi ba · Yatagan ·
Types of daggers	Anelace · Applegate–Fairbairn fighting knife · Arkansas toothpick · BC-41 · Bagh nakh · Baselard · Bichawa · Bollock dagger · Cinquedea · Dirk · Ear dagger · Emeici · Facón · French Nail · Hachiwara · Hunting dagger · Janbiya · Kaiken · Kalis · Kard · Katar · Khanjar · Kris · Mark I trench knife · Ocean Edge Knife · Parrying dagger · Poignard · Push dagger · Seme · Shobo · Sica · Stiletto · Fairbairn-Sykes Fighting Knife · Tantō · Marine Raider stiletto · V-42 stiletto · Yarara Parachute Knife · Yoroidōshi ·
Knife manufacturers	Aitor · Al Mar Knives · American Tomahawk Company · Bear MCG · Benchmade · Böker · Buck Knives · Caber · Camillus Cutlery Company · Cattaraugus Cutlery Company · Chris Reeve Knives · Cold Steel · Columbia River Knife & Tool · Cuisinart · Cutco Cutlery · DOVO Solingen · Dexter-Russell · Elkhorn · Ek Commando Knife Co. · Emerson Knives · Erizo · F. Dick · Fällkniven · FAMAE · Fiskars · Füritechnics · Gerber Legendary Blades · Global · Golok · Hanwei · Imperial Schrade · J. A. Henckels · KA-BAR · Kershaw Knives · KitchenAid · Korin Japanese Trading Company · Kyocera · Leatherman · Mad Dog Knives · MercWorx · Microtech Knives · Morseth · Murphy Knives · Ontario Knife Company · Opinel · Pro-Tech Knives · Randall Made Knives · Ranz · Rigid Knives · Rösle · SOG Specialty Knives · Sabatier Aîné & Perrier · Spyderco · Strider Knives · Survival Aids · TEKNA · Thiers Issard · Victorinox · W. R. Case & Sons Cutlery Co. · Walther arms · Wenger · Western Knife Company · Wilkinson Sword · Windlass Steelcrafts · Wüsthof · Yarara Ltd ·
Knifemakers	A.G. Russell · American Bladesmith Society · Rex Applegate · James Black · Blackie Collins · John Nelson Cooper · Ernest Emerson · Jerry Fisk · Phill Hartsfield · Bill Harsey, Jr. · Gil Hibben · Knifemakers' Guild · Zanjan Knifemakers · Jimmy Lile · Bob Loveless · Bob Lum · William F. Moran · Ken Onion · Ralph Osterhout · Walter Doane "Bo" Randall, Jr. · Chris Reeve · Jody Samson · William Scagel · Robert Terzuola · Michael Walker · Buster Warenski · Daniel Winkler ·
	Book:Knives and daggers · Category:Knives / Daggers ·
	Switzerland portal

Categories: Camping equipment | Mechanical hand tools | Military equipment of Switzerland | Pocket knives | Swiss inventions | Victorinox | Wenger

Thread: Aspect ratio problem with mencoder / mplayer

Thread

28th Aug 2010 11:19 #1

wipp
Member
Join Date: Jun 2010
Location: Switzerland

I'm trying to cut some commercials in some h264 avi files. Sometimes the P frames aren't exactly at the beginning of the commercial part, so I cut and reencode the surrounding part (using avidemux). This way I limit the quality loss to this part.

Then I rejoin the avi files using mencoder. The resulting video looks fine, at least up to mplayer SVN-r29796-4.4.3.
But with mplayer SVN-r30554-4.4.3, the reencoded parts of the video play with a different aspect ratio than the rest (not reencoded), i.e. mplayer changes the aspect ratio during the playback.

Since I don't want to reencode all my files, I'm searching for a method to get rid of the aspect ratios stored in the files.

Code:
```
mencoder -ovc copy -oac copy -force-avi-aspect 16:9 -o outfile infile
```

doesnt help, even if I split the avi, apply this command to each segment and rejoin it.
The funny thing is that i can set the correct aspect ratio for each single segment. But after joining them, their "original" / wrong aspect ratios come back.

Is somebody familiar with this behaviour?

Quote

28th Aug 2010 12:35 #2

ricardouk
Member

Join Date: Mar 2005
Location: Portugal

i think you can change aspect ratio after with MPEG4 Modifier

have you tried removing the force-avi-aspect 16:9 from the script and let mencoder do that on its own, i think it detects aspect ratio, fps, resolution without the user specifying it.

try another player that doesnt use mplayer, mpc-hc or vlc

I love it when a plan comes together!
Ricardo Santos

Quote

28th Aug 2010 14:11 #3

wipp
Member
Join Date: Jun 2010
Location: Switzerland

Thanks for your reply!

> i think you can change aspect ratio after with MPEG4 Modifier

I can't test MPEG4 Modifier since I have to create a mp4 first and my MP4Box reports that my raw videos are corrupt (importing them into a mkv file works).
By the way: if I put the merged video in a mkv the same change of aspect ratio occurs (even if I first put all parts in different mkv with correct aspect ratio and then merge them).

> have you tried removing the force-avi-aspect 16:9 from the script and let mencoder do that on its own, i think it detects aspect ratio, fps, resolution without the user specifying it.

Leaving out the -force-aspect-ratio option has no effect on the output file (at least mplayer keeps changing the aspect ratio).

> try another player that doesnt use mplayer, mpc-hc or vlc

vlc doesn't change the aspect ratio. It plays the video in 4:3 if I leave it as an avi, but it plays it in 16:9 if I put it in a mkv container.

Quote

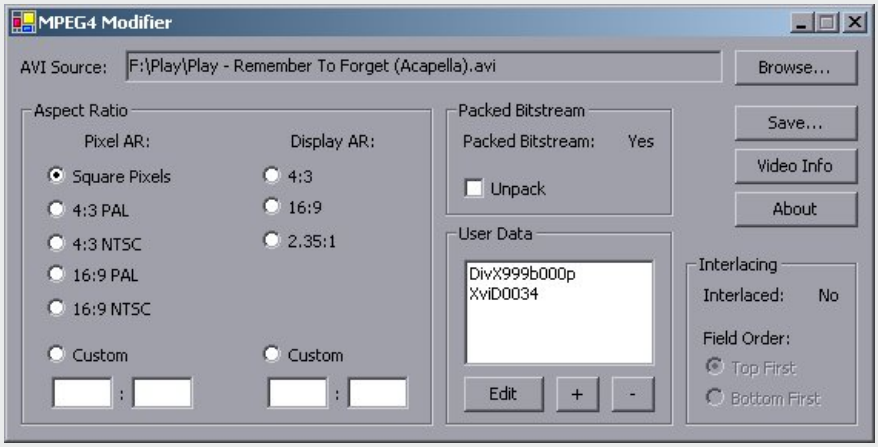
28th Aug 2010 14:21 #4

ricardouk
Member

Join Date: Mar 2005
Location: Portugal

> Originally Posted by **wipp**
> I can't test MPEG4 Modifier since I have to create a mp4 first

you can open avi files with it



I love it when a plan comes together!
Ricardo Santos

Quote

28th Aug 2010 14:29 #5

ricardouk
Member

Join Date: Mar 2005
Location: Portugal

another solution would be to convert the commercials to DV and then use virtualdub to edit/join at will without reconverting.

Code:
```
ffmpeg -i video.mkv -target pal-dv video.dv
```

1- convert each video to dv and put in a folder called DV
2- cut edit each video from folder DV with virtualdub and save it again (without reconverting it) to folder EDITED
3- join all dv files from folder EDITED with virtualdub and save it (no recompression) to Folder FINAL
4- convert the video from the FINAL folder to avi with virtual dub

Last edited by ricardouk; 28th Aug 2010 at 14:36.

I love it when a plan comes together!
Ricardo Santos

Quote

28th Aug 2010 14:31 #6

wipp
Member
Join Date: Jun 2010
Location: Switzerland

OK, but with my file or with one segment, I get "This is not a valid MPEG-4 video (startcode not found at beginning of frame). Codec: H264."

Quote

28th Aug 2010 14:44 #7

ricardouk
Member

> Originally Posted by **wipp**
> so I cut and reencode the surrounding part (using avidemux). This way I limit the

> quality loss to this part.

Join Date: Mar 2005
Location: Portugal

ah i thought you were converting to avi (xvid+mp3) i don't normally associate avi with h264...

try to convert them to dv (minimal quality loss) and then edit/join with virtualdub and then convert the final dv to your favourite format, with the DV format you can cut and join without worrying about special keyframes.

I love it when a plan comes together!
Ricardo Santos

Quote

28th Aug 2010 17:22 #8

wipp
Member
Join Date: Jun 2010
Location: Switzerland

OK, thanks. I'll give this a try.

Quote

+ Reply to Thread

Quick Navigation Editing Top

Similar Threads

MPlayer aspect ratio By nic3 in forum Software Playing	Replies: 4 Last Post: 10th Jan 2011, 13:13
Mplayer pipe output to Mencoder possible ? By neapandele in forum Video Conversion	Replies: 1 Last Post: 28th Oct 2009, 03:43
MPLAYER pipe to MENCODER possible? By neapandele in forum Newbie / General discussions	Replies: 0 Last Post: 27th Oct 2009, 06:56
Having to reload mpeg2enc/mencoder/mplayer each start By SeveralButchersApron in forum ffmpegX general discussion	Replies: 5 Last Post: 31st Aug 2008, 11:41
mencoder and mplayer packages By mattr in forum ffmpegX general discussion	Replies: 1 Last Post: 30th Dec 2007, 09:05

« **Previous Thread** | **Next Thread** »

6.6. Rescaling movies

Often the need to resize movie images emerges. The reasons can be many: decreasing file size, network bandwidth, etc. Most people even do rescaling when converting DVDs or SVCDs to DivX AVI. If you wish to rescale, read the Preserving aspect ratio section.

The scaling process is handled by the `scale` video filter: `-vf scale=width:height`. Its quality can be set with the `-sws` option. If it is not specified, MEncoder will use 2: bicubic.

Usage:

```
mencoder input.mpg -ovc lavc -lavcopts vcodec=mpeg4:mbd=2:trell \
    -vf scale=640:480 -o output.avi
```

6th January 2011, 08:47 #1 | Link

Nonoman
Registered User

Join Date: Apr 2008
Posts: 1

mencoder resizing using -sws flag

Hy there!

I want to downscale a FullHD stream to pal-dvd size 720x576 using the -sws flag.
(walking on a Ubuntu 10.04 server)

My commandline looks like this:

Code:

```
mencoder video-delay3,6.mpg -channels 6 -oac copy -ovc lavc -of mpeg -mpegopts format=dvd:tsaf
-vf scale=720:437,expand=720:576,harddup -sws 9 -srate 48000 -af lavcresample=48000
-lavcopts threads=256:vcodec=mpeg2video:vrc_buf_size=1835:vrc_maxrate=9800:vbitrate=9799:
keyint=3:trell:mbd=2:precmp=2:subcmp=2:cmp=2:dia=2:predia=2:cbp:mv0:vqmin=1:lmin=1:
dc=10:aspect=16/9:vstrict=0:vpsize=500:vfdct=0:idct=0 -ofps 25 -o movie.mpg
```

So, this is using lanczos for resizing. (-sws 9)

How do I adjust the strength of the filter?

man (manual of) mencoder tells me:
scale [= w: h [: ilaced [: chr_drop [: par [: par2 [: presize [: noup [: arnd]]]]]]]]

- Scales the image with the software (slow) and performs a color space conversion between YUV and RGB through (see the -sws).

<par> [: <par2>] (also see -sws)
Set some scaling parameters depending on the scaling, which was selected with-sws.

- -sws 2 (bicubic): B (blurring) and C (Gain)
- 0.00:0.60 Standard
- 0.00:0.75 "precise bicubic" VirtualDub
- 0.00:0.50 Catmull-Rom spline
- 0.33:0.33 Mitchell-Netravali spline
- 1.00:0.00 cubic B-spline
- -sws 7 (gaussian): sharpness (0 (soft) - 100 (sharp))
- -sws 9 (Lanczos): filter length (1-10)

But how do I have to use this in command-line?

my regards

Last edited by Nonoman; 7th January 2011 at 09:05. Reason: Edited text

Quote

15th January 2011, 10:44 #2 | Link

www669
Registered User

Join Date: Feb 2010
Posts: 28

can you tell me what's the mencoder websit?

Quote

15th January 2011, 10:44 #3 | Link

www669
Registered User

Join Date: Feb 2010
Posts: 28

do you have mencoder's options!
i can't find it

Quote

4th July 2011, 22:42 #4 | Link

rogerdpack
Registered User

Join Date: Jun 2011
Posts: 34

Ok I figured it out for mplayer:
http://betterlogic.com/roger/2011/07/mplayer-lanczos/
basically
-sws 9 -vf scale=1280:0:0:0:10# 10 is the filter length...I think so anyway, though it doesn't complain if I set it to 11.

Also does anybody know if it accepts sharpen parameters, like -ssf ls=75.0 -ssf cs=75.0 ?

if it works, it will display "Lanczos" in the output after a few lines.

Quote

Post Reply

Tags

command-line, lanczos, mencoder, resize, ubuntu

« Previous Thread | Next Thread »

Posting Rules

You **may not** post new threads
You **may not** post replies
You **may not** post attachments
You **may not** edit your posts

BB code is **On**
Smilies are **On**
[IMG] code is **On**
HTML code is **Off**

Forum Rules

Forum Jump

MPEG-2 Encoding Go

All times are GMT +1. The time now is 10:33.

Doom9.org - Archive - Top

6.3. Encoding two pass MPEG-4 ("DivX")

The name comes from the fact that this method encodes the file *twice*. The first encoding (dubbed pass) creates some temporary files (`*.log`) with a size of few megabytes, do not delete them yet (you can delete the AVI or rather just not create any video by redirecting it into `/dev/null` or on Windows into `NUL`). In the second pass, the two pass output file is created, using the bitrate data from the temporary files. The resulting file will have much better image quality. If this is the first time you heard about this, you should consult some guides available on the net.

Example 6.2. copy audio track

Two pass encode of the second track a DVD to an MPEG-4 ("DivX") AVI while copying the audio track.

```
mencoder dvd://2 -ovc lavc -lavcopts vcodec=mpeg4:vpass=1 -oac copy -o /dev/null
mencoder dvd://2 -ovc lavc -lavcopts vcodec=mpeg4:mbd=2:trell:vpass=2 \
    -oac copy -o output.avi
```

Example 6.3. encode audio track

Two pass encode of a DVD to an MPEG-4 ("DivX") AVI while encoding the audio track to MP3. Be careful using this method as it may lead to audio/video desync in some cases.

```
mencoder dvd://2 -ovc lavc -lavcopts vcodec=mpeg4:vpass=1 \
    -oac mp3lame -lameopts vbr=3 -o /dev/null
mencoder dvd://2 -ovc lavc -lavcopts vcodec=mpeg4:mbd=2:trell:vpass=2 \
    -oac mp3lame -lameopts vbr=3 -o output.avi
```

H.264 encoding - CPU vs GPU: Nvidia CUDA, AMD Stream, Intel MediaSDK and x264 - BeHardware

>> Graphics cards

Written by Guillaume Louel

Published on April 28, 2011

URL: http://www.behardware.com/art/lire/828/

Page 1

Introduction

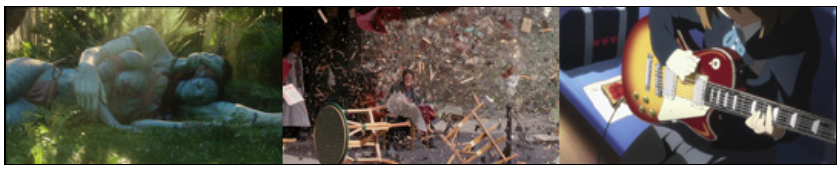
Over recent years, graphics card manufacturers have been highlighting the ability of their GPUs to handle more than just gaming graphics. Although there has been some success in the market for High Performance Computing (financial services sector and so on), the concept of GPGPU (General Purpose computing on GPU) is still struggling to find purchase with the general consumer. Standards like OpenCL and DirectCompute are working to make an impact but so far this is still fairly minor.

Video, naturally!

General consumer GPGPU usage is nevertheless quite common when it comes to video encoding solutions. On paper, the idea seems a very good one. GPU power can already be drawn on for video decoding (Blu-rays for example), via a dedicated circuit integrated into the GPU and it would therefore seem logical to use GPUs for encoding too.

After a fairly limited start with the first version of Badaboom in 2008, transcoding is back on the agenda with Windows 7, which includes the possibility of using your graphics card to transcode files automatically towards a peripheral (portable video player, smartphone, as long as it is compatible and detected by Windows 7). Many free or paid applications have set to getting to grips with file transcoding, whether these files be destined for mobile peripherals, tablets or games consoles.

A common point of all these applications is that they often use libraries designed by GPU manufacturers themselves. Thus NVIDIA supplies an H.264 encoder developed in CUDA (nvcuvenc), AMD has its own which uses Stream technology (AMD Media Codec package at the bottom of the page), and even Intel have put in their penny's worth, as we saw at the beginning of the year on the launch of the Core Sandy Bridge processors with its MediaSDK (which is itself partly based on the IPP library).



But what do all these solutions give in practice? Is all the software that uses them equivalent? What about encoding speed and quality? And how are these encoding solutions to be measured against what is the reference when it comes to H.264 encoding, the open source software x264 (that we regularly use in our CPU tests)? We'll be getting to grips with all these burning questions in this report!

Before going further, we want to thank Nicolas et fils for the loan of hardware used in the tests, as well as Jason Garrett-Glaser (x264 developer) for replying to our questions on H.264. If you wish to go into more depth on this subject, here are the links to some of the resources used in researching this article:

- The H.264 recommendation
- The book The H.264 Advanced Video Compression Standard by Iain Richardson (the site for which makes available some interesting free content here)
- Jason Garrett-Glaser's blog

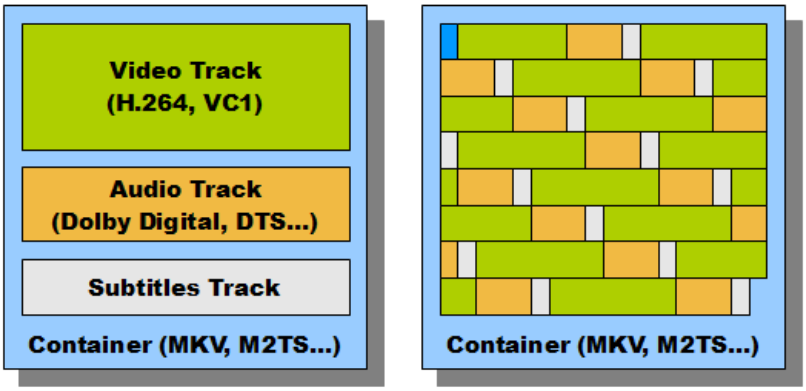
Page 2

Container, codec, transcoding

Before really getting to grips with our subject and to help understand exactly what we're comparing, let's go back over a few points with respect to the subject of video.

Container, codecs

What is wrongly called a video file is above all else a container. The concept is simple, it interlaces within it the various contents of the file, namely the video track, the audio track and potentially the subtitling tracks.



Left: view of principle. Right: interlaced tracks.

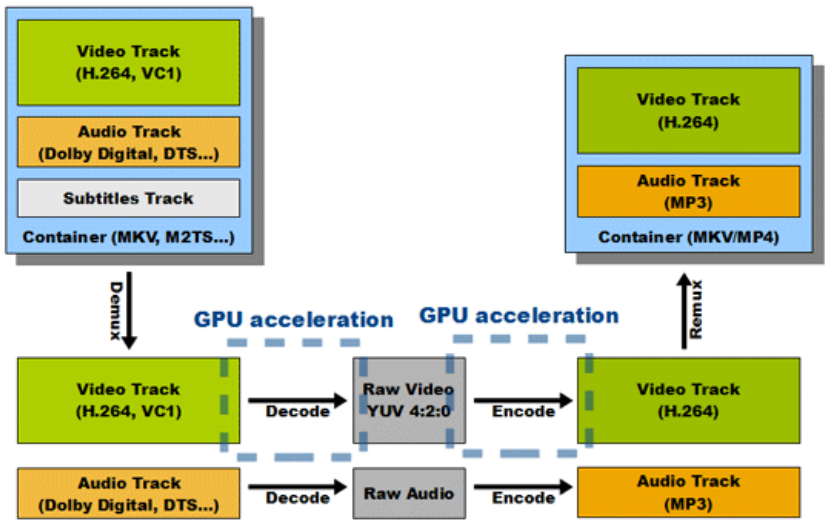
Interlacing the content facilitates the simultaneous playback of different tracks without requiring too much movement within the file. Related data (the audio that corresponds to an image for example) thus remains in close proximity. This is particularly important for video discs (DVDs, Blu-rays) or when transmitting a digital flow (via digital broadcast for example).

There isn't really any relationship between a container and the format of the tracks it contains. An AVI file can contain video tracks encoded in many formats (MPEG-1, Xvid and so on), with the same going for audio (MP3, AAC and so on). A utility like MediaInfo will tell you exactly which tracks make up a file, and which format they are encoded in. There are also some technical limitations, which can for example prevent a video format such as H.264 from being integrated (straightforwardly) in an AVI file.

Transcoding

Therefore, when we talk about transcoding solutions, we're actually talking about changing one or several of the formats in the original file. The compression formats can be changed to give a smaller file, converting, say, a DVD (VOB container, MPEG-2 video, Dolby Digital audio) to a video file (AVI, XviD, mp3). Sometimes you might want to retain the same video format but reduce the size of the files, by, for example, converting a Blu-ray (.m2ts, H.264, DTS-HD) into a file that can be played on a tablet (.mp4, H.264, AAC) or games console. Even if you don't change the video format (H.264 on each side), you might want to recompress the video, either to change its size (Blu-rays take up a lot of space), screen size (reduce to 1280 x 720 instead of 1920 x 1080) or specificity of the destination device (iPads, for example, can't read all H.264 compression profiles).

There are, then, a large number of stages to manage when it comes to transcoding a file, and not all these stages can be accelerated by the GPU. In the diagram below, from left to right, you can see the stages necessary for the transcoding of a video file:



Click to enlarge.

Of all these stages, only two can currently be accelerated by a GPU, decoding the original video track to a raw video format (using dedicated GPU circuits, the same that are used during accelerated playback of a DVD or a Blu-ray) and encoding (using either the GPU processing units [CUDA – NVIDIA / Stream – AMD] or a part of the GPU dedicated to this task [Intel method for Sandy Bridge/HD 3000].

Only these two stages can be accelerated then, but they are by far the most resource hungry. Here is, for example, a breakdown of an encode that we carried out for our tests (file extracted from a Blu-ray to an MKV 720p file):

	Time (in seconds)
Demux (eac3to)	83
Audio transcode (BeSweet)	17
Video transcode (x264, first pass)	394
Video transcode (x264, second pass)	4506
Remux (mkvmerge)	59
Total	5059

This is a very high quality encode that we processed to create a source file (we'll come back to this). Video encoding time is therefore significant. To conclude on the subject of transcoding, note two points it's important to bear in mind for the rest of this article:
- Decoding and encoding of video are tasks that are carried out in parallel, frame by frame. In theory encoding takes longer than decoding, but this isn't always so in certain cases with respect to GPU acceleration.
- Although some tools allow you to break down the time taken for each stage, not all the software that we tested does so. When we talk about transcoding times further on, this will therefore consist of the time taken for all stages (demux, video encoding, audio, remux) and not only video encoding time!

Let's now move on to the specificities of H.264.

Page 3

H.264 (1/2)

H.264

Sometimes known as AVC (often when talking about Blu-rays) or MPEG4 Part 10 (ISO terminology), H.264 (ITU terminology) is a video compression standard often considered to be the most technically advanced compression format. In ISO terminology it follows MPEG-1 (CD video), MPEG 2 (used on DVDs) and MPEG 4 Part 2 (XviD).

It's a standard with an open specification (downloadable here), the use of which is subject to a certain number of patents belonging to various big names in computing, electronics, telecommunications and various universities (Apple, Cisco, France Telecom, Fraunhofer, Microsoft, Mitsubishi, Panasonic, Philips, Sony and Toshiba to name but a few; the full list of patents is available here). All these companies have come together to create a pool of patents managed by MPEG LA. In the framework we're looking at (personal use, non-commercial), no license is required.

H.264 is now widely used, whether this be for Blu-rays (where it coexists with a Microsoft format, VC-1, which also comes under the supervision of patents handled by MPEG LA) or because of the fact that H.264 decoding can be accelerated by most current general consumer devices (smartphones, tablets, games consoles and so on). It is also widely used on online video sites, something that we noted in our test of the AMD Brazos platform when the Flash software posed a problem in terms of video acceleration.

Video compression in practice

Without going into too much detail, we're going to give you a general overview of how H.264 works, which will allow us to put into context any issues we have with the various encoders tested further on in the article.

A video is a stream of frames, compressed one after the next. H.264, like many formats before it, subdivides frames into blocks of pixels (squares of between 4x4 and 16x16 pixels, known as macroblocks). To achieve the final result, an encoder must determine the data required for the recreation of a frame (this is what's known as prediction; there are two types) then compress this data as efficiently as possible (here again, there are two distinct types of compression).

Temporal prediction

Behind this term lies a very simple concept: a succession of frames in a video are very likely to be linked, to have a relationship to each other. Whether the sequence is made up of a static shot or moving characters, or a camera pan taking in a landscape, the successive frames will very often have numerous links between them. Starting from this principle, the encoder will attempt to identify blocks in the destination frame taken from one or several previous frames. Differences may be minuscule, down to a quarter of a pixel. This is what's known as motion estimation and is a type of processing for which GPUs are likely to be very efficient. Henceforth we will refer to temporal prediction as inter prediction.

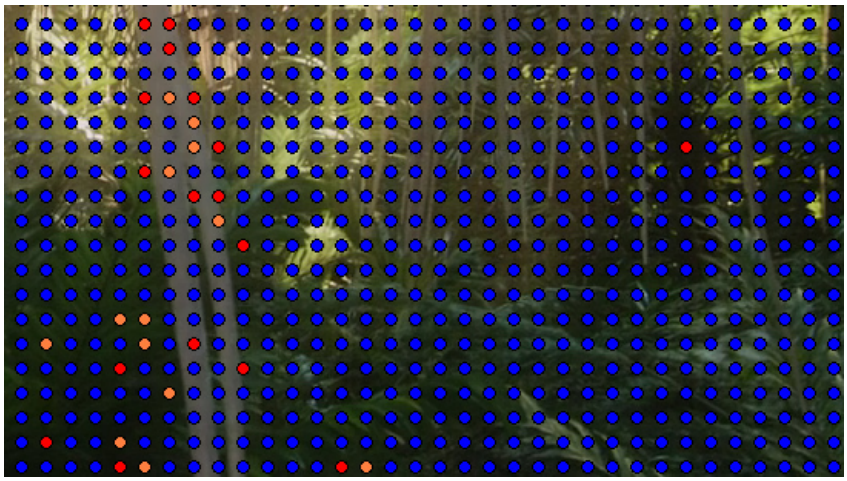
Avatar, Fox Pathé Europa

On this frame taken from a scene in Avatar (using Elecard StreamEye) where the camera moves left to right, while advancing, you can see that the elements of the frame which are at different depths (the creeper on the left, the ferns at the bottom, the lighter coloured ferns at the top and the creepers at the top) all move differently within the frame.

Spatial prediction

Again this term is more complicated than it sounds. Spatial prediction consists in compressing data from the current frame. Rather than looking for similarities between current and previous frames, spatial prediction looks for similarities within the current frame. This concept is equivalent to the compression of static frames (JPEG and so on) and we will refer to this as intra prediction in the rest of this article.

Of course, an encoder can use both types of prediction within the same frame. If we take the same frame as before (still using Elecard StreamEye):



The inter predicted blocks (temporal prediction, using a previous frame) are in blue/yellow and the intra predicted blocks (spatial prediction, within the current frame) in red/orange. Avatar, Fox Pathé Europa

Note that even when an encoder does its best to be as precise as possible in its predictions, they aren't necessarily perfect. In every case of prediction where an attempt is made to find similarities between one block and another (whether within the same frame, intra prediction, or in another, inter prediction), the destination doesn't always exactly resemble the source. This isn't a problem as the difference can simply be made up. By definition, it will be small as, after all, the blocks resemble each other.

Compress what and how?

Our predictions generate two types of quite distinct data. On the one hand precise data such as motion vectors used in predictions. If the encoder has put in the effort to obtain precision down to a quarter of a pixel, obviously you don't want to compress this data and lose accuracy. For such essential data, H.264 uses lossless coding (entropy coding). There are two distinct types, CAVLC and CABAC. The difference between the two is that the second requires considerably more processing, whether to encode or decode. You may remember that some time back, the GeForce 8800s offered decoding of accelerated H.264 streams, though this was partial. There was no GPU support for CABAC decoding at the time (this was added to the following generations). CAVLC is less resource hungry but also less efficient (lower compression ratio).

For data which can be subjected to a (slight) loss, such as, for example, the residual image that corresponds to the difference between the source and destination blocks during a prediction (see above), a destructive compression, known as quantization, is applied.

The combination of this compressed data (lossless and lossy) makes up your H.264 video file.

Now that we've set down a general overview of H.264, we're going to look in detail at a few of the points that came to the fore during our tests.

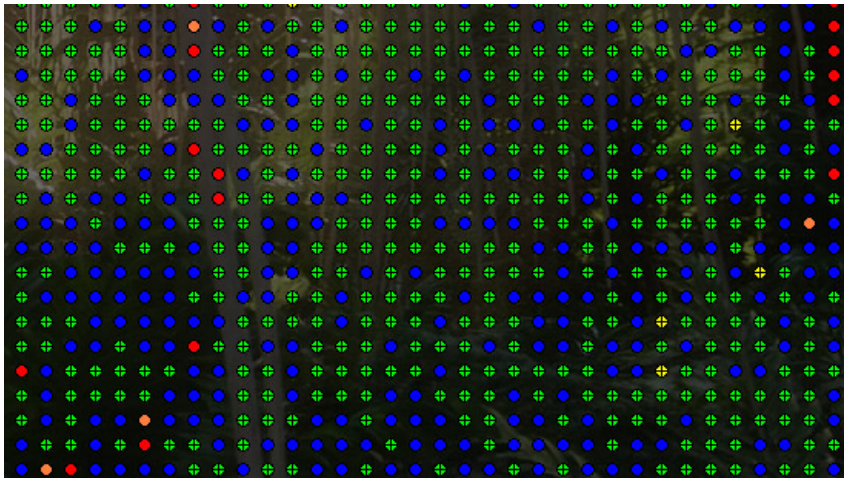
Page 4

H.264 (2/2)

B-frames

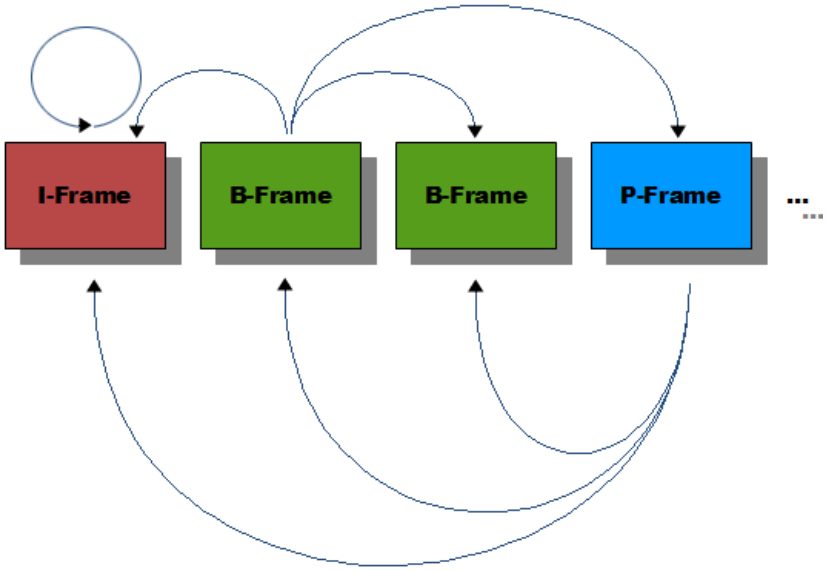
As we have seen, there are two possible types of prediction: inter prediction – the prediction of a frame from earlier frames – or intra prediction – prediction from the current frame. In practice, you'll find different types of frame within a video stream:
- some frames result only from intra prediction. Such a frame can be decoded independently of any other. This is what's known as a key frame. Key frames serve as reference points and are vital for features such as fast forward and rewind. In practice a key frame should be placed on average at least once every ten seconds. These frames, which result entirely from intra prediction are known as I-frames.
- Other frames result from a combination of intra prediction and blocks which refer to earlier frames (inter prediction). These are known as P-frames. The frame presented on the previous page is a P-frame.
-H.264 adds a third (optional) type of frame. Rather than referring only to earlier frames, a B-frame may also refer to a later frame! This does of course make encoding somewhat more complex, but being able to refer to both previous and forthcoming frames limits the number of intra predicted blocks (bigger) that have to be added. This therefore improves compression!

Here's a B-frame which combines three types of blocks:



In blue, the inter predicted blocks (temporal prediction, referring to a previous frame), in orange/red, the intra predicted blocks (spatial prediction, within the current frame). In green, the blocks referring to later frames! Avatar, Fox Pathé Europa

The following diagram summarises the different frame possibilities:



Deblocking filter

One of the recurrent problems of formats that use block coding concerns the sharp edges that can appear between blocks. If the blocks are overly compressed using approximate similarities, the edges of the macroblocks can sometimes be made out. You have probably noticed this on overly compressed JPEG frames:

Examples of overly compressed blocks

H.264 has a deblocking filter that can be applied to each macroblock. The encoder defines whether and how this filter is to be applied by the decoder. This parameter can require significant processing and may be put to one side by some encoders with a view to gaining time, as we will see.

Adaptive quantization matrices

H.264 quantization (a lossy compression technique) can be adapted by applying different values to each block. These are known as independent quantization matrices, or simply adaptive quantization. Instead of applying a uniform compression across all the residual data (subtraction of the prediction from the macroblock), some blocks are given priority over others, which means more detail can be retained in complex parts of the frame.

Profiles

Some features such as those previously described, can be particularly resource hungry, whether in terms of encoding or decoding. As H.264 aims to be a universal format, its authors have defined multiple usage profiles which either allow or disallow the use of certain features of the format. A mobile phone, for example, can only support the most basic profile, while a Blu-ray will use the most advanced mode. We're interested in the three modes used by the various encoders: Baseline, Main and High. Here's a little summary of the advanced features they support:

Profiles	Baseline	Main	High
Deblock. filter	X	X	X
B-frames		X	X
CABAC		X	X
Adaptative quant			X

In practice we'll always go for the highest possible profile, according to the formats supported by the destination peripheral. A games console like the Playstation 3 supports all three profiles, whereas the iPad only officially supports the first two. Before starting the coding process, it's important to bear in mind the capacities of the peripheral you'll be playing your file on: when you're decoding H.264 on a specific machine, the format isn't as universal as you might think!

Page 5

Measuring quality: PSRN, SSIM and the pitfalls

Measuring quality: PSRN, SSIM and their drawbacks

Here at Hardware.fr / Behardware.com, we try and design our tests to be as objective as possible. It's fairly easy to determine performance when the result can be easily measured (processing time for a given task for example). When it comes to video, the question of objective judgement of video quality is unfortunately very… subjective. The true objective criteria is visual quality of the video as perceived by the human eye. This is something that unfortunately can't be measured or quantified other than by human tests.

Over the years, several tools have been developed to try and compare the quality of one video to another. The basic concept remains the same: frames from the compressed video are compared one by one to the source and this gives us a series of values for each of the frames that make up the video.

The standard measure (or metric) used to compare two frames is PSNR which attempts to determine the level of corrupting noise in a compressed image in comparison to the source. Used above all to judge static frame compression formats, PSNR is considered to be a very statistical meaure depending largely on the compression format chosen or the particularities of the encoder. The other metric used is called SSIM and attempts to determine the structural similarity between frames, with the aim of being a bit more realistic than PSNR.

While in practice SSIM is a better indicator of visual quality, the problem remains relatively complex as human perception is difficult for an algorithm to measure. Our eyes are for example instinctively attracted to faces. This means that the human eye will prefer an image on which the face is sharp but which may be otherwise inaccurate (and which has a low PSNR), to an image with a more even quality across the areas but in which the face isn't as well defined (but ironically a higher PSNR!).

Added to this problematic is the fact that as with any metric for which the algorithm is known, it's very easy to optimise an encoder for one algorithm or another to the detriment of the overall video quality. The x264 encoder illustrates this issue quite well: as well as having options that allow you to optimise the encoder for a film or cartoon, you can also optimise it to get the highest possible PSNR and SSIM values! Here's a little example of what this can give in a scene from the film Inception. We have calculated the average PSNR and SSIM values with four different x264 optimisations (none, Film, PSNR, SSIM). Note that the PSNR values are expressed in dBs (the higher the value the better the sigal to noise ratio), while the SSIM is a value between 0 and 1 which indicates correlation to the source image, with 1 showing perfect correlation.

	Film	No tune	PSNR	SSIM
SSIM	0.9670	0.9674	0.9641	0.9696
PSNR	38.0043	38.0223	37.7941	38.1465

Attempting to optimise an algorithm doesn't always work depending on the scene. The scene chosen here is full of explosions and PSNR optimisation gives the lowest scores. The SSIM metric seems to be best, obtaining the two best scores for the SSIM and PSNR. Is this borne out when we look at the frames? Let's see what we get on a static frame taken from each video:

[No Tune] [Film] [PSNR] [SSIM]

Move the mouse over/click on the links to view the corresponding frame. Inception, Warner Bros

Let's start with the most obvious. On the PSNR frame, not much is right. Entire parts of the face are blurry and the shading on the right is blocked. Is the SSIM version better than the Film version? No. The eyelashes are sharper on the Film version and there's more detail in the face. So why does it have a lower score when the SSIM version is slightly more blurred? This is because of a parameter which makes PSNR and SSIM comparisons even more complex, the psycho visual optimisations which try to retain a maximum of details in interesting areas. This optimisation can be made out in a static frame but becomes even clearer in a video series: the video quite simply retains more details and what look like artefacts on a static frame (for the PSNR and SSIM metrics) actually give improved quality though with lower metric scores.

For these reasons, while we will give the SSIM/PSNR scores for the various encoders, these scores cannot be considered to be of absolute value!

Page 6

Number of passes, dynamic GOP

In addition to the format specificities discussed above, encoders can also be configured with certain specific settings. Here are some explanations of the role of these options.

One or two passes?

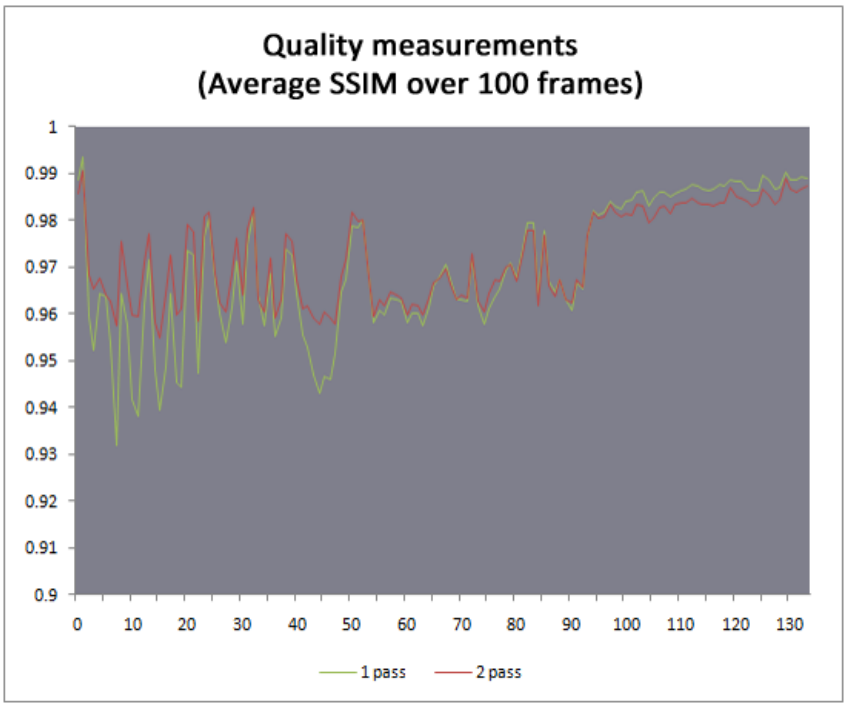
When you want to compress a video, you're generally limited to a given file size that you don't want to exceed, for example for storage reasons (fitting the video onto a DVD or a maximum of videos in a smartphone and so on).

The usual method is to specify the bitrate – an average amount of data per second of encoding – so as to guide the encoder. Transcoding software generally handles this issue back to front – you indicate the desired file size and the encoder calculates the bitrate. However the bitrate serves simply as a guide and limit not to be exceeded. The encoder will then have to decide for itself where it can economise and where it can go over its 'budget'.

The problem of this approach is that an encoder doesn't know how your video is constituted. Are all the scenes equally complex? Can the end of the video be compressed more than the beginning? It's difficult to say without viewing it first. This is why when you want to obtain a good level of quality, you use two passes. This means that the encoder carries out two passes over the video and can better manage its bitrate.

We have compressed two videos with identical quality settings, using first one and then two passes to illustrate the problematic. Every 100 frames (around 4 seconds), we calculated the average bitrate per frame, as shown in the graph below.

Hold the mouse over the graph to compare changes in quality.



As you can see in green, with a single pass the encoder is conservative and doesn't go either too high or too low at any point. In red, with two passes, the encoder chooses to budget higher at the beginning of the video and less at the end. If you hold the mouse over the graph, you can see why by looking at the visual quality as compared with the SSIM metric. The last minutes of the video can be more highly compressed without compromising what is a very high level of similarity to the source, higher than on the rest of the video. By using a second pass, you can see that the quality remains more constant throughout the video, which is exactly what we're looking for: jumps in quality make viewing uncomfortable.
While the advantage of using two passes has been recognised for several years now, all the GPU encoders that we have tested here make do with a single pass. This is a shame as implementing a second pass would be a very simple way of improving quality significantly.

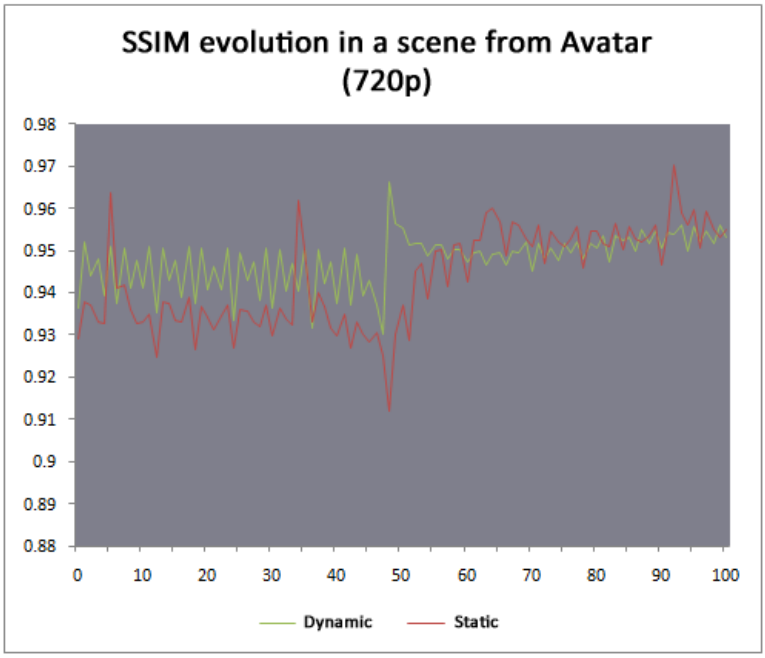
Dynamic GOP

As we have seen, encoders have three types of frame at their disposal (I-frame, B-frame, P-frame), which can be used as they see fit. A good encoder will place a key frame (I-frame) when, for example, there's a change in scene. What about poor encoders? They use a fixed frame structure (known as GOP, Group of Pictures)! They simply place an I-frame every 29 frames. Here's an illustration of what two encoders give in practice:

```
Encoder 1: IPPPPPPPPPPPPPPPPPPPPPPPPPPPIPPPPPPPPP
Encoder 2: IPPPPPPPPBBBPBBBPBBBPBBBPBBBPBBBPPPPPI
```

Using a static GOP is among the worst ideas and has disastrous consequences in terms of quality!

If you look at the graph of one of our quality metrics, you can already guess what visual defects are going to appear:



The first problem of this 29 frame packet structure is the jumps in quality. Each I-frame restores similarity but quality drops with each P-

frame that follows! Look at what happens on frame 48 however. Here there's a scene change. The dynamic encoder inserts a key frame to mark the change and give a good level of quality. However, the static encoder waits for its cycle to come round and tries to restore quality with each P-frame, gradually improving quality until things return to normal with the next I-frame!
The difference in sharpness is there for all to see:

Hold the mouse over the image to see the next I-frame.

Avatar, Fox Pathé Europa

Now that we have gone over the potential pitfalls, let's move on to the tests.

Page 7

Test scenes, configuration

Test scenes

We used files from three Blu-rays:

- Avatar Extended Version, *James Cameron, Fox Pathé Europa*
- Inception, *Christopher Nolan, Warner Bros*
- K-On!!, *Kyoto Animation, Pony Canyon*

Avatar has the advantage of combining filmed and computer generated scenes. Originally developed for 3D, there isn't too much fast movement, which is something encoders find easier to handle. The scene lasts 9 minutes and 20 seconds.

Avatar, Fox Pathé Europa

The scene from Inception is almost epileptic. The camera is static but the scene full of explosions. This presents many interesting challenges from the point of view of the encoders. Another challenge here is that Inception is a Blu-ray encoded in VC1.

Inception, Warner Bros

For K-On!! we encoded the first episode of the disc. Animation uses big tinted coloured areas which pose very different problems than those of standard films.

K-On!!, Kyoto Animation, Pony Canyon

We used these raw files, without recompression for our 1080p encoding tests. For software other than x264, we tried to set quality to a maximum as far as possible – we'll come back to this for each encoder as quality options often vary a great deal. A high bitrate was chosen for this compression: 10 Mb/s (4.5 GB/hour), which is a little higher than digital HD (around 7/8 Mb/s). A high bitrate facilitates the task in terms of compression. This is still quite some way off a Blu-ray as the bitrates for our test files were 24.2, 25.8 and 35.4 Mb/s respectively. This scenario corresponds to a re-encoding for a console.

We also wanted to test a second encoding scenario at the lower resolution of 720p. So as not to disadvantage any of the encoders, we reduced and recompressed our 1080p sources beforehand so as to obtain new 720p source files. We carried out this operation at x264 in slowest mode with a bitrate of 25 Mb/s. Each piece of software may, in effect, use very different resizing methods which would have prevented accurate comparison of frames from an identical source. By removing this resizing element (which in any case doesn't take up much time in terms of the encoding process) and starting with a very high quality source, we reduced as far as possible any false differences that may exist.

Note that we deactivated any options which try to retouch colours to make the videos brighter (increase of saturation and so on), whether this be in the software or, where the source image is decoded by the GPU, in the GPU control panels. In addition to being useless, they would have disadvantaged the encoders in the SSIM and PSNR tests.

Test configuration

We used the following configuration:

- Intel Core i7 2600K
- MSI H67MA-E45 (1155)
- 4 GB Crucial DDR3 1333 MHz
- Windows 7 64 bits

In addition we used three AMD and NVIDIA graphics cards (equivalent in price terms) to see if the most powerful models gave any advantage in terms of quality, or a reduction in encoding time:

- Radeon HD 5750
- Radeon HD 6850
- Radeon HD 6970
- GeForce GTS 450
- GeForce GTX 460
- GeForce GTX 570

For the Intel QuickSync solution, we tested the HD 3000 included in the 2600K.

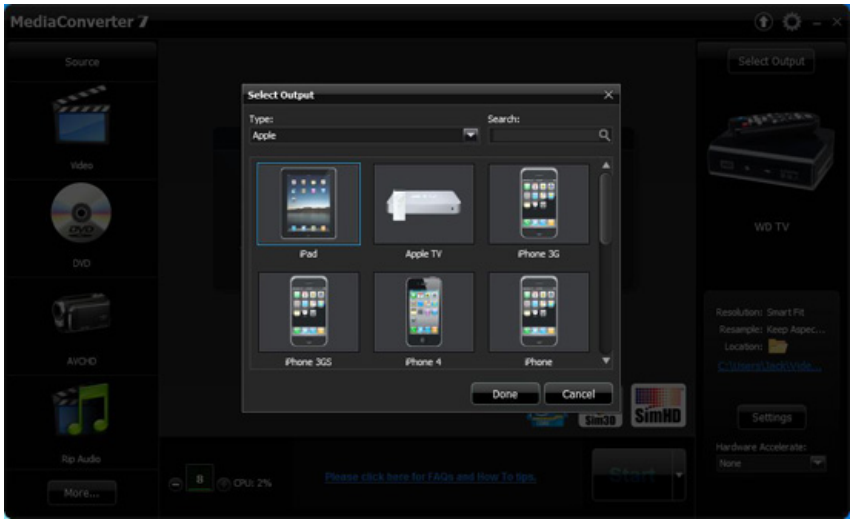
A final word on the presentation of the results. So as to make the results legible, the graphs on the following pages, as well as the comparisons of frames, require the use of an HTML5 compatible Internet browser. Most of you will already be using one, but if not, you can choose to install any of the following browsers:

- Mozilla Firefox 4
- Google Chrome 10
- Opera 11
- Apple Safari 5
- Microsoft Internet Explorer 9

ArcSoft Media Converter, Avatar 720p (1/4)

ArcSoft Media Converter, Avatar 720p (1/4)

Developed by ArcSoft (authors of the Blu-ray Total Media Theater playback software), Media Converter 7 (version 7.1.15.55, abbreviated as AMC in our graphs) proposes to transcode sources either in the form of video files or DVDs. The software has profiles for consoles (Xbox, PS3, Wii, PSP), tablets (iPad), smartphones and other peripherals designed for TV playback (Apple TV, WD TV, etc).



We created two profiles manually for the 720p and 1080p resolutions, maximizing quality options. The H.264 high profile is not supported and Arcsoft seems to have custom encoders for the CUDA and Stream versions. Quality is almost constant whichever GeForce is used, with a very slight advantage with the GeForce GTX 460. In practice, this advantage makes no difference. Note also that the GeForce GTX 570 (too recent?) wasn't detected by our version of Media Converter. Quality was strictly identical across the various Radeons.

Let's start with the results at 720p for Avatar. To recap, our files encoded at 720p were carried out with a requested video throughput of 4 Mbit/s.

Avatar 720p

Arcsoft	Profile	CABAC	Ref Frames	Bitrate	B Frames	Dynamic GOP
CPU	Main	Yes	2	3913	Yes	No (I80, 1P2B)
GeForce	Main	Yes	2	3953	Yes	No (I80, 1P2B)
Radeon	Baseline	No	2	4076	No	Yes* (I50)
HD 3000	Main	Yes	2	3947	Yes	No (I80, 1P2B)

Arcsoft	%I	%P	%B	SSIM	PSNR
CPU	1,26	33,75	65,00	0,963824	37,40571
GeForce	1,26	33,75	64,99	0,797308	29,72937
Radeon	2,51	97,49	-	0,953717	37,20051
HD 3000	1,25	33,75	65,00	0,96165	37,80678

The first surprise was that using a Radeon drastically changes the encoding options. It uses the Baseline profile and there are no B-frames or CABAC. You do however get virtually dynamic GOP here in as much as it uses an I-frame on scene changes. The others are more radical: one I-frame ever 80 frames, then between them a repeated group of 1 P-frame followed by 2 B-frames. If you compare the scores of the Arcsoft encoders between themselves, the CPU version is quite some way ahead for SSIM. The Intel MediaSDK seems to be optimised for PSNR (and therefore optimised for benchmarks rather than visual quality). The CUDA version gives an extremely low score, which requires some explanation.

Arcsoft	Time
CPU	122
AMD HD 6970 full	297
AMD HD 6850 full	299
AMD HD 5750 full	254
NV GTX 570 full	X
NV GTX 460 full	110
NV GTS 450 full	149
HD 3000 full	135

In terms of processing time, the pure CPU version is devilishly fast, while the Radeon version, in spite of the baseline profile, is twice as slow. The Radeon HD 5750 is faster than the bigger Radeon models, though without any obvious reason.

The average SSIM values (or PSNR) do not however mean a great deal. As with any relationship, it's how the difficult times are managed that provides us with the parameters for judging encoder quality. To highlight these difficult situations more clearly, we isolated two series of 500 distinct frames, taken from our extract.

Use an HTML 5 compatible browser to see the graph!
Click here to see the PSNR graph of this scene.

Note that for this graph and for those that follow, we had to go for a dynamic scale in accordance with the content. So as to help you to visualise the differences between the various graphs, an orange line shows an SSIM of 0.9. You can also increase the size of the graph by clicking on the icon at top right.

Our first extract of 500 frames (scene 1) can be broken down into three parts. From around frame 0 to around 157, we have a relatively static scene. The main character is filmed in the pod and only the head moves. This is very easy to compress and all the encoders do a good job here. The scene ends with a fade to black (which gives us a score of 1 as all encoders know how to encode an entirely black image!). Up to frame 362 a new scene is introduced, more complex than the previous one as the camera moves in a slow tracking shot from left to right while advancing slowly. At frame 363, a new scene appears, again a relatively slow moving scene.

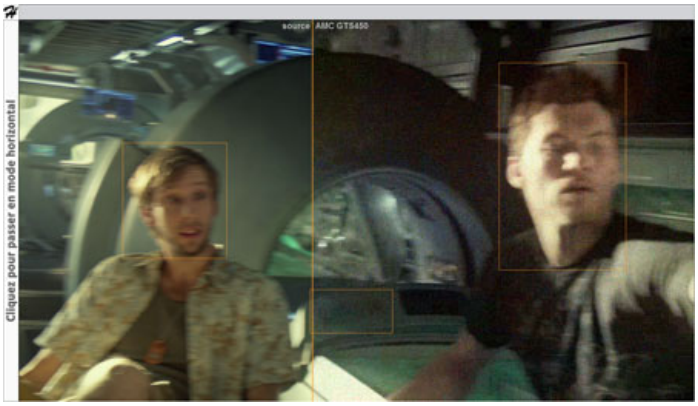
The Radeons use the least advanced encoding technique (baseline profile), but use dynamic GOP. The result of this is that while the quality of the encoding is systematically down on the others, the use of dynamic GOP means the Radeon gives better quality than the HD 3000 when there's a change in scene. The GeForces are however completely lost when there's a change in scene and there's a notable loss in quality. Much more serious than this however, when the image changes a little too much (and remember this is a slow-moving scene…), the Arcsoft Cuda encoder loses its footing between two I-frames (the peaks you can see every 80 frames) with a significant loss in quality. These jumps in quality are very visible. So as to show you what this gives in practice, we have isolated an identical frame from each encoded sequence, around half a second after the change in scene. The frames used for our comparison below are stored in PNG format so as to prevent any deformation. They may take a few seconds to load depending on the speed of your connection:

Click here to display the frame comparison in a new tab.

While, as we said, the GTX 460 CUDA encoding has a very slight advantage when it comes to conservation of textures over the GTX 450 version, in practice neither are useable because of the excessive artefacts. The Arcsoft CUDA encoder is quite simply not up to the task. Our other encoders show differences in sharpness. The HD 3000 encoder lags behind from a visual point of view, while the AMD versions and CPU encoder are comparable, with the AMD encoder even seeming slightly better when you look closely at the texture of the skin. In practice these two frames are both a long way off the source frame unfortunately, with a significant loss of detail.
Our second scene includes several quite rapid movements and uses motion blur heavily, which facilitates the task of the encoders. What does this give in practice?

Use an HTML 5 compatible browser to view the graph!
Click here to see the PSNR graph for this scene.

The Arcsoft encoders function well with motion blur. The CUDA version is too poor to speak of, but our three other encoders are neck and neck. The minute SSIM nuances are however visible. Look at the second frame that we have extracted from a motion blur sequence:



Click here to display the frame comparison in a new tab.

If we only take faces into account, the AMD encoder does pretty well and conserves a maximum of sharpness on the faces. There are however artefacts in the shaded areas between the dark and light areas on the faces on the CPU and HD 3000 encoder versions (left part of Jake's forehead, on the right of the screen). Let's see if this level of quality is confirmed in the slightly more complex scenes!

Page 9

ArcSoft Media Converter, Inception/K-On!! 720p (2/4)

ArcSoft Media Converter, Inception 720p

Now let's move on to the film Inception, still at 720p, and the short extract we have chosen of just 40 seconds, which does however include some particularly interesting explosions.

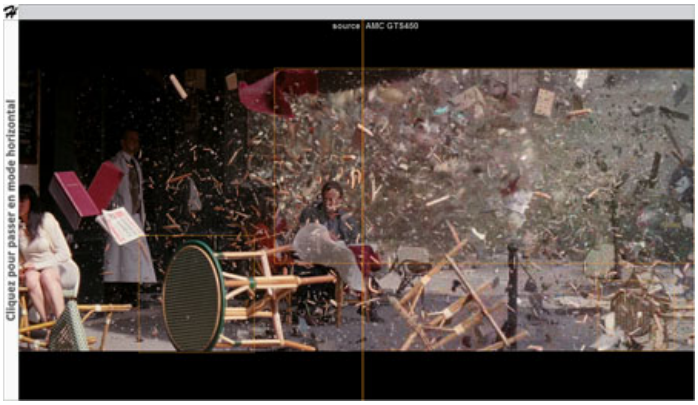
Arcsoft	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
CPU	Main	Yes	2	3567	Yes	No (I80, 1P2B)	0,970119
GeForce	Main	Yes	2	3506	Yes	No (I80, 1P2B)	0,903284
Radeon	Baseline	No	2	4072	No	Yes* (I50)	0,956287
HD 3000	Main	Yes	2	3744	Yes	No (I80, 1P2B)	0,957889

When you look at the SSIMs obtained across the whole length of the scene, the Arcsoft CPU encoder seems to stand out a bit more from the Radeon/HD 3000 versions, which are at a similar level. Given the limited length of the extract, we won't give encoding times here.

We extracted a scene of 500 frames. During the first 400, shots with explosions alternate with shots of characters (51, 192, 352) before the scene finishes with a much calmer sequence.

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

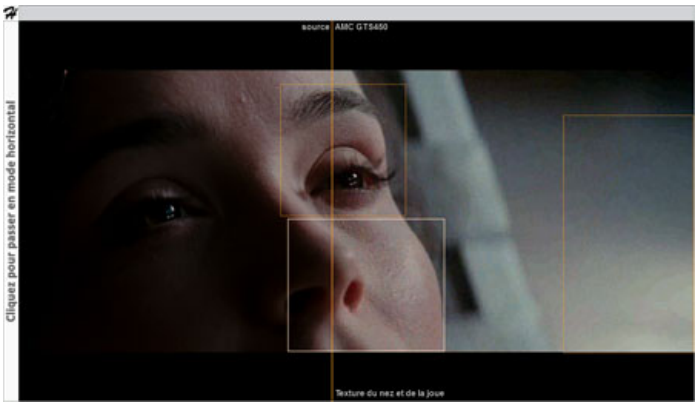
The advantage of the Arcsoft CUDA encoder is that it allows you to detect scene changes easily: at each trough in quality, there's a scene change! The Radeon encoder is a good way down on the Intel encoder in the explosion scenes, while the Intel is behind in the static scenes. Let's check out what this translates to visually, firstly in an explosion frame:



Click here to display the frame comparison in a new tab.

We're going to put discussion of the quality of the CUDA version to one side and concentrate on the Radeon. While in the slow-moving Avatar scenes, the AMD encoder held its own better, here it's a long way behind. This isn't really a surprise as the baseline profile is a lot less efficient than the others in terms of compression! Although the CPU version of Arcsoft preserves more detail, the result is far from great.

What about the end of the sequence where the scene is more static?



Click here to display the frame comparison in anew tab.

None of the encoders maintain the grain in the background. The Intel encoder is the least precise on the face and the cheek and the contour of the eye are particularly blurred. This is difficult to forgive when you think that your eye is drawn to this part of the image first.

Now for our last 720p test sequence.

K-On!! 720p

We encoded an entire episode (24 minutes 11 seconds) of this animation, with a relatively static image. At 4 Mbit/sec, this shouldn't be too much of a challenge for our encoders.

Arcsoft	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
CPU	Main	Yes	2	4104	Yes	No (I80, 1P2B)	0,992123
GeForce	Main	Yes	2	4102	Yes	No (I80, 1P2B)	0,981309
Radeon	Baseline	No	2	2687	No	Yes* (I50)	0,989955
HD 3000							

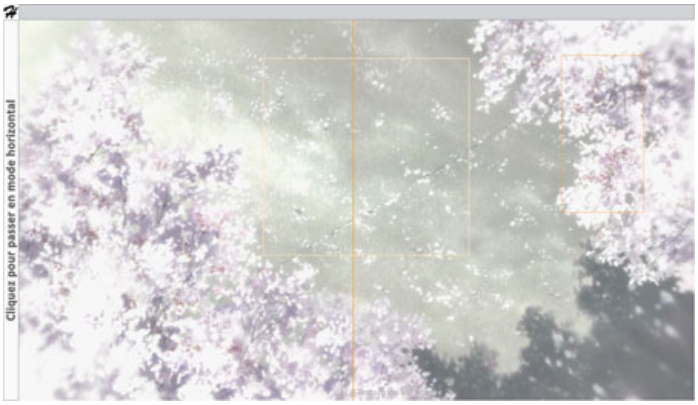
The first surprise is that the Arcsoft Intel encoder didn't work here (the software crashed). The second surprise was that the score with the Arcsoft CUDA encoder, while still behind the Radeons, seemed to indicate that the result wouldn't be the usual soup!

Arcsoft	Time
CPU	264
AMD HD 6970 full	338
AMD HD 6850 full	338
AMD HD 5750 full	349
NV GTX 570 full	X
NV GTX 460 full	231
NV GTS 450 full	256
HD 3000 full	X

In terms of processing time, the CPU version was once again very fast, even if the Arcsoft CUDA encoders were faster.

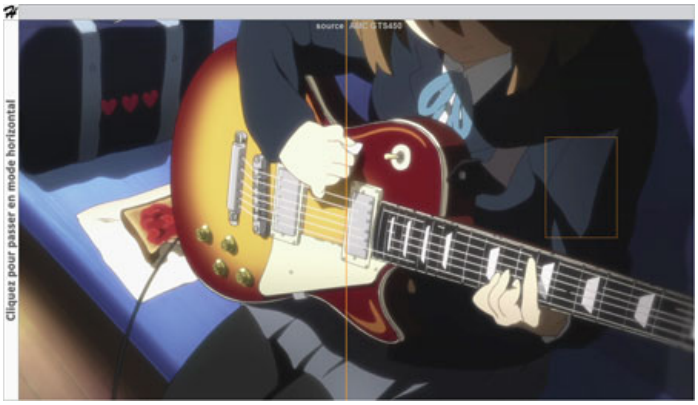
Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

The GeForce encoder still struggles with scene changes but the result still seems excellent. What does this give in practice?



Click here to display the frame comparison in a new tab.

This isn't a mistake, the GeForce version is actually the one that conserves most detail! With the CPU and Radeon versions of the encoding, you lose all the grain. As this frame is static for several seconds, these two encoders have deliberately chosen to blur the scene, which is to be regretted! This is a tendency that we have already noticed and which also shows, once again, the pitfalls of relying on SSIM and PSNR type readings!



Click here to display the frame comparison in a new tab.

In a scene with fewer textures and more solid colours, the differences are minimal, with just a few slight artefacts. Nothing too concerning.

Let's now see if moving up to 1080p and a higher bitrate changes things for the Arcsoft application.

ArcSoft Media Converter, Avatar 1080p (3/4)

ArcSoft Media Converter, Avatar 1080p

Here we use the same scenes as before, the source files being the Blu-ray video files.

Arcsoft	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
CPU	Main	Yes	2	9787	Yes	No (I80, 1P2B)	0,694692
GeForce	Main	Yes	2	9325	Yes	No (I80, 1P2B)	0,834388
Radeon	Baseline	No	2	10200	No	Yes* (I50)	0,965323
HD 3000	Main	Yes	2	9899	Yes	No (I80, 1P2B)	0,969538

The SSIM average for the CPU version of the Arcsoft encoder is excessively low. This is because this encoder uses a variable frame rate for 1080p encoding. While our source uses a constant frame rate at 23.976 frames/second, the Arcsoft encoder varies the frame rate depending on the complexity of the scene. While it's often question of a few milliseconds, here the differences are more significant (via MediaInfo):

Original frame rate: 23.976 fps
Minimum frame rate: 8.108 fps
Maximum frame rate: 29.132 fps

The result is that after around 1500 frames, the encoder applies a frame rate that is almost a quarter of the original frame rate on a scene it sees as being slow-moving and static. SSIM/PSNR comparisons become impossible!

The use of VFR is a valid technique as it's included in the H.264 spec. It is however generally reserved for very precise usage, such as streaming or surveillance cameras. In our eyes, it doesn't make any sense to use it to encode films as the playback of VFR files can be problematic on some software/peripherals, even more so at 1080p where the file is destined for playback on a TV. The CUDA/Stream/Media SDK encoders are not confronted with this problem and use a constant frame rate.

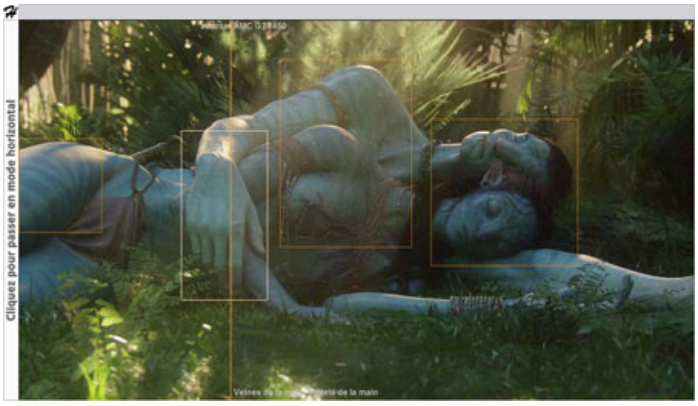
Arcsoft	Time	cpu%	Watts
CPU	1417	99	108
AMD HD 6970 full	1403	26	128
AMD HD 6850 full	1403	28	106
AMD HD 5750 full	1401	31	102
NV GTX 570 full	X	X	X
NV GTX 460 full	1403	49	190
NV GTS 450 full	1403	48	168
HD 3000 full	1403	15	69

In terms of encoding time, MediaConverter seems to stumble when it comes to the audio track, which is encoded in parallel to the video track. This takes an abnormally long time.

Nevertheless let's look at the changes in the SSIM over the first 500 frames of the scene:

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

During the first 500 frames, the frame rate doesn't change for the Arcsoft CPU encoder. The Intel HD 3000 encoder is hot on its heels. What does this give visually?



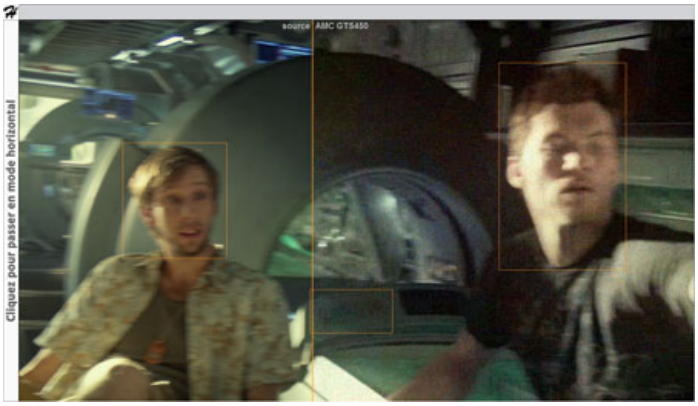
Click here to display the frame comparison in a new tab.

In spite of an average score above, the MediaSDK version isn't very good, with a lack of sharpness in the faces and skin textures in comparison to the CPU and Radeon versions.

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

In this second scene the CPU version is already 7 frames behind, falsifying any comparison and making the scores obtained on the CPU

version useless. For the other encoders, the results with the MediaSDK and Stream versions seem very good, with a small advantage for the MediaSKD encoder. Let's check on the frames:



Click here to display the frame comparison in a new tab.

Again, quite ironically, the Radeon version conserves more detail. You can see this clearly by looking at the section in the middle of the screen (completely blurred by the other encoders).

Page 11

ArcSoft Media Converter, Inception/K-On!! 1080p (4/4)

ArcSoft Media Converter, Inception 1080p

Arcsoft	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
CPU	Main	Yes	2	9536	Yes	No (I80, 1P2B)	0,977691
GeForce	Main	Yes	2	8677	Yes	No (I80, 1P2B)	0,92732
Radeon	Baseline	No	2	10100	No	Yes* (I50)	0,969574
HD 3000	Main	Yes	2	9178	Yes	No (I80, 1P2B)	0,968505

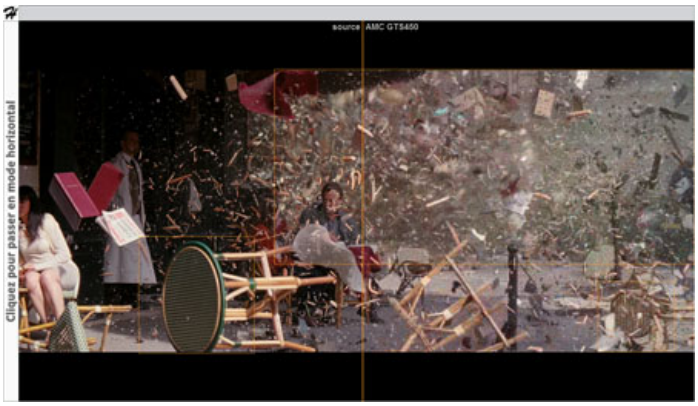
While the Arcsoft encoder still uses a variable frame rate, in practice it's as if it didn't (via MediaInfo):

Original frame rate: 23.976 fps
Minimum frame rate: 23.976 fps
Maximum frame rate: 23.976 fps

Let's look at the results of the various encoders included in ArcSoft Media Converter:

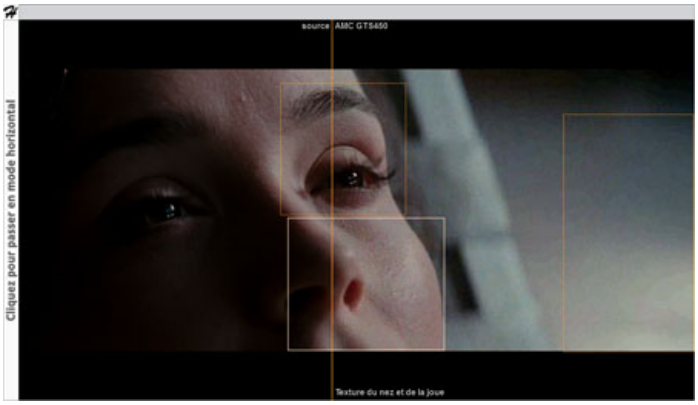
Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

The CPU version seems to do best whether in the explosion scenes or the calmer ones. Is this borne out in practice?



Click here to display the frame comparison in a new tab.

Even if the result is far from perfect, the CPU encoder does best in terms of conservation of detail. The Radeon encoder is literally nowhere, the baseline profile having exceeded itself!



Click here to display the frame comparison in a new tab.

On more static frames, you can once again see the loss of sharpness on the face with the Intel MediaSDK encoder. This is particularly visible on the forehead. There's a loss in grain in the background texture and increased artefacts. The Radeon encoding is of comparatively better quality, quite close to the quality with the CPU encoder.

K-On !! 1080p

Arcsoft	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
CPU	Main	Yes	2	10300	Yes	No (I80, 1P2B)	0,977691
GeForce	Main	Yes	2	10200	Yes	No (I80, 1P2B)	0,92732
Radeon	Baseline	No	2	6336	No	Yes* (I50)	0,969574
HD 3000	Main	Yes	2	9879	Yes	No (I80, 1P2B)	0,968505

Once again the Arcsoft CPU encoder uses a variable frame rate (via MediaInfo):

Frame rate mode Variable
Frame rate: 23.976 fps
Minimum frame rate: 7.992 fps
Maximum frame rate: 47.952 fps

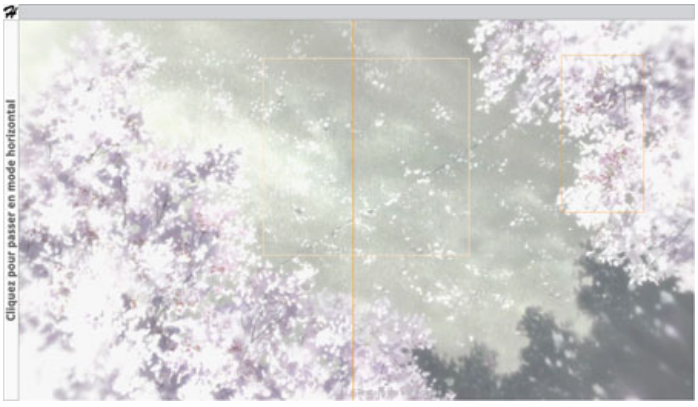
The final 11 frames from the original video are missing which makes the scores for this encoder invalid. Let's nevertheless look at the results on the scene we've isolated:

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

The scores are relatively high, but as we saw in practice on the 720p version, this doesn't mean very much. Will this result in a blurred image once again with our encoders?

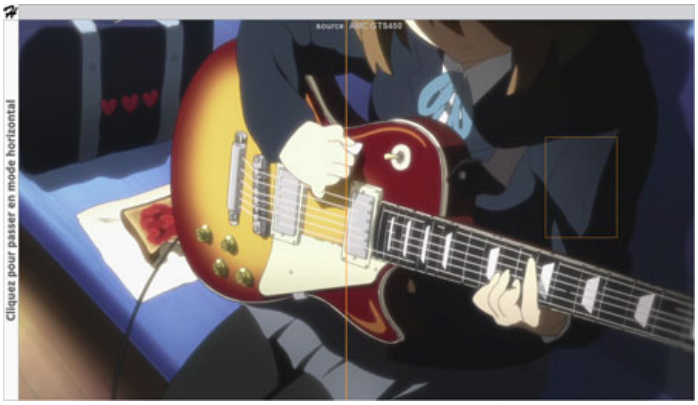
Arcsoft	Time
CPU	478
AMD HD 6970 full	590
AMD HD 6850 full	676
AMD HD 5750 full	673
NV GTX 570 full	X
NV GTX 460 full	413
NV GTS 450 full	575
HD 3000 full	460

In terms of encoding time, there's an enormous difference from one card to another here, and this time the Radeon HD 6970 is fastest.



Click here to display the frame comparison in a new tab.

You can find better! Once again the Arcsoft CUDA encoder does well on this scene, in which it conserves the grain best. The Radeon Stream encoder renders an artistic blur while the CPU and MediaSDK versions are slightly better, without being perfect.



Click here to display the frame comparison in a new tab.

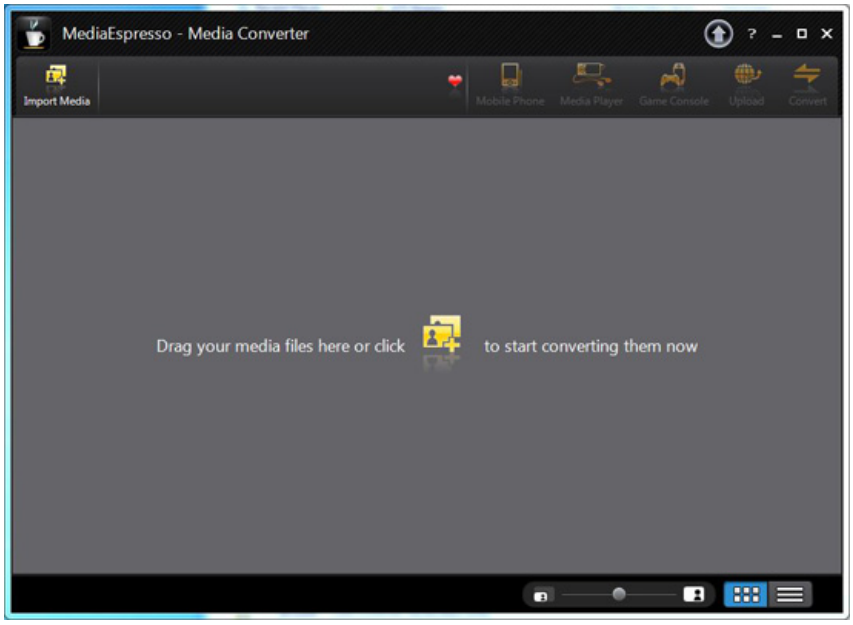
The reason for the weaker scores for the CUDA encoder becomes clearer when you look carefully at the lines. The line above our point of interest disappears slightly, which is what makes the difference. There are still some small artefacts on the solid colours, though we'd be happy with such a result in all of our tests!

Page 12

Cyberlink MediaEspresso, Avatar 720p (1/4)

Cyberlink MediaEspresso, Avatar 720p (1/4)

From the developers of PowerDVD, Cyberlink MediaEspresso (in version 6.5.1229, abbreviated to CME in our graphs) is also part of the video transcoding brotherhood. This application has quite a friendly interface. At the top of the screen, you'll find the access to the presets for smartphones, media players, games consoles and Internet sites.



For our tests we created two profiles for 720p and 1080p, maximising all available quality settings. The good news is that the Cyberlink application allows you to choose to turn GPU acceleration on for any of the decoding or encoding phases, or both (see our explanation on page 2).

Avatar 720p

Cyberlink	Profile	CABAC	Ref Frames	Bitrate	B Frames	Dynamic GOP
CPU	Baseline	No	2	3925	No	No (I29)
GeForce (Decode)	Baseline	No	2	3924	No	No (I29)
GeForce (Encode)	Baseline	No	1	4000	No	No (I29)
GeForce (Full)	Baseline	No	1	4000	No	No (I29)
Radeon (Decode)	Baseline	No	2	3925	No	No (I29)
Radeon (Encode)						
Radeon (Full)						
HD 3000 (Decode)	Baseline	No	2	3925	No	No (I29)
HD 3000 (Encode)	Baseline	No	2	4000	No	No (I29)
HD 3000 (Full)	Baseline	No	2	4000	No	No (I29)
HD 3000 (Decode)	Baseline	No	2	3000	No	No (I29)
HD 3000 (Encode)	Baseline	No	2	3000	No	No (I29)
HD 3000 (Full)	Baseline	No	2	3000	No	No (I80)

Cyberlink	%I	%P	%B	SSIM	PSNR
CPU	3,45	96,55	-	0,962377	37,21243
GeForce (Decode)	3,45	96,55	-	0,961343	36,65205
GeForce (Encode)	3,45	96,55	-	0,948857	35,82225
GeForce (Full)	3,45	96,55	-	0,948257	35,5811
Radeon (Decode)	3,45	96,55	-	0,962381	37,21054
Radeon (Encode)					
Radeon (Full)					
HD 3000 (Decode)	3,45	96,55	-	0,962387	37,21104
HD 3000 (Encode)	3,45	96,55	-	0,956934	37,24237
HD 3000 (Full)	3,45	96,55	-	0,948853	36,32375
HD 3000 (Decode)	3,45	96,55	-	0,948059	35,6577
HD 3000 (Encode)	3,45	96,55	-	0,946059	35,96769
HD 3000 (Full)	1,26	98,74	-	0,940224	35,21177

The good news stops pretty quickly however when you look at how the software handles encoding. Firstly, it only supports the baseline profile and as we saw with the Arcsoft software, this isn't likely to improve quality. The baseline profile is highly disadvantaged at an equal bitrate in action scenes. Next, the bitrates are surprising. The Cyberlink CUDA and MediaSDK encoders use a constant bitrate, namely exactly 4 Mb/second all the time. Such a strategy doesn't favour quality (remember why using two passes improves quality!). Next, the 'Quick' mode, available for encoding with the HD 3000 (abbreviated by the letter Q in our graphs below), reduces the bitrate automatically. Lastly, the GOP is static and, in performance terms, the CPU encoder is once again ahead of the rest! Note, the Radeons crashed systematically when we tried to encode our test scene with GPU acceleration. We reproduced this problem with the AMD AVIVO encoder that is used here.

Cyberlink	Time
CPU	222
AMD HD 6970 decode	177
AMD HD 6970 encode	X
AMD HD 6970 full	X
AMD HD 6850 decode	177
AMD HD 6850 encode	X
AMD HD 6850 full	X
AMD HD 5750 decode	181
AMD HD 5750 encode	X
AMD HD 5750 full	X
NV GTX 570 decode	192
NV GTX 570 encode	144
NV GTX 570 full	165
NV GTX 460 decode	192
NV GTX 460 encode	145
NV GTX 460 full	165
NV GTS 450 decode	192
NV GTS 450 encode	145
NV GTS 450 full	165
HD 3000 decode/qual	178
HD 3000 encode/qual	149
HD 3000 full/qual	140
HD 3000 decode/fast	104
HD 3000 encode/fast	144
HD 3000 full/fast	120

In terms of encoding time, we note that graphic card decoding accelerates CPU encoders but slows GPU encoders down! The encode versions are faster than the Full versions…

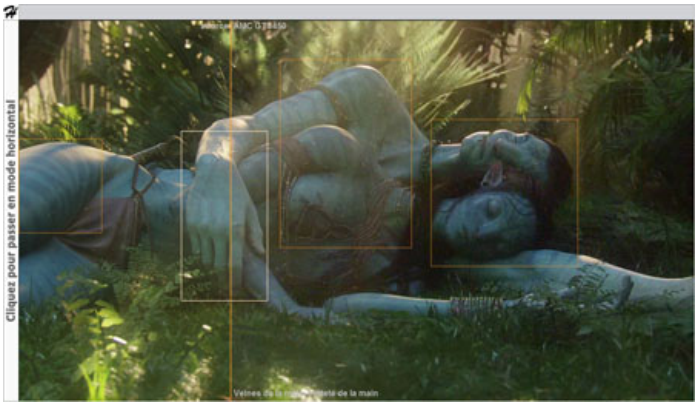
Let's check all this in practice in our first scene in Avatar.

Use an HTML5 compatible browser to view the graph!

Click here to view the PSNR graph of this scene.

As you can see, the scale of our graph has been drastically reduced: The reason for this is the Intel encoders fall off very abnormally in the Full version. The reason for these troughs is fairly easy to understand:

It seems likely that it's a result of a bug, especially as the artefacts don't appear either in the purely decode version or the purely encode version with the HD3000! If you click on both these encodes in our graphs, the scale returns to normal. As we might have expected, if you only use the decode part of the GPU acceleration, all the encoders perform at an identical level overall, with however a very small variation between the GeForces and the rest. The three encoders (CPU, CUDA encode, MediaSDK encode) suffer from the lack of dynamic GOP when there's a change in scene around frame 363. Let's check this visually:



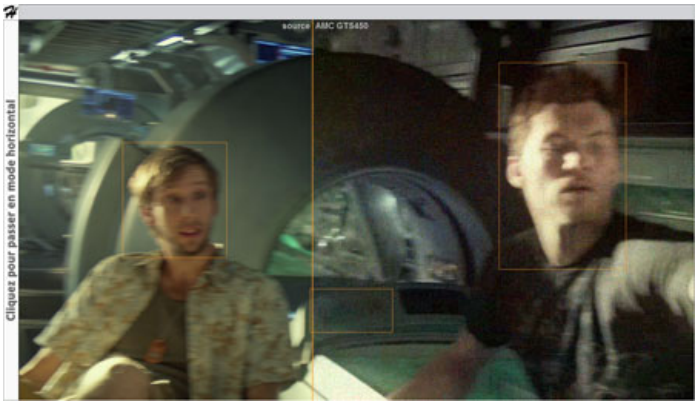
Click here to display the frame comparison in a new tab.

Everything is very blurred here, especially with the HD 3000 encoder when you look at the details of the skin, the vegetation or quite simply the faces. Ten frames after a change in scene, the lack of dynamic GOP and detection of new scenes creates unbearable jumps in quality. The CUDA version of the Cyberlink encoder doesn't have the same soup-like aspect as the Arcsoft one, which is a bonus!

Now let's move on to our second scene with all the motion blur.

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

There are no unusual troughs here and the results seem homogenous, even if the CUDA encoder is a small notch down on the others. Let's check this visually:



Click here to display the frame comparison in a new tab.

The results are quite poor in terms of sharpness (and therefore conservation of detail) for all cases. Note the artefacts above the grille, particularly on the GeForce/HD 3000 versions. Now let's see how Cyberlink does in the other scenes.

Page 13

Cyberlink MediaEspresso, Inception/K-On!! 720p (2/4)

Cyberlink MediaEspresso, Inception 720p

Moving on to the film Inception, still at 720p from which we have selected a short extract of just 40 seconds but which includes some particularly interesting explosions.

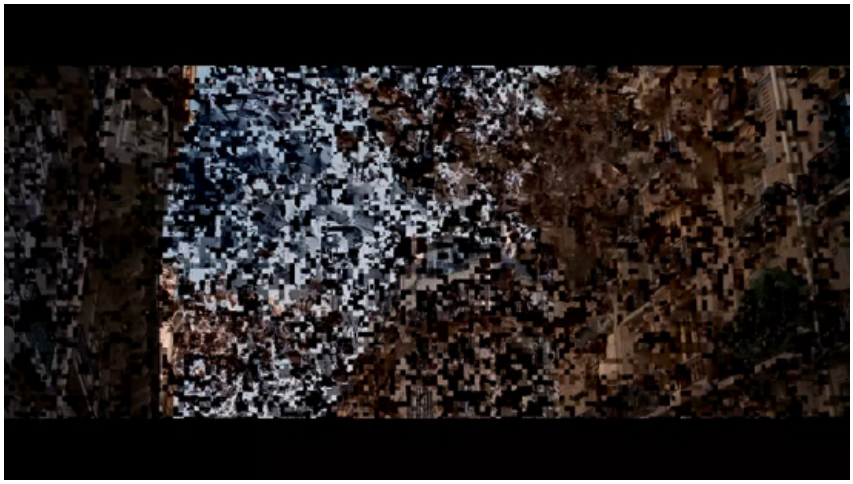
Cyberlink	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
CPU	Baseline	No	2	3776	No	No (I29)	0,965825
GeForce (Decode)	Baseline	No	2	3759	No	No (I29)	0,964494
GeForce (Encode)	Baseline	No	1	4000	No	No (I29)	0,949086
GeForce (Full)	Baseline	No	1	4000	No	No (I29)	0,948404
Radeon (Decode)	Baseline	No	2	3772	No	No (I29)	0,965898
Radeon (Encode)	Baseline	No	2	3963	No	No (I29)	0,951156
Radeon (Full)	Baseline	No	2	3963	No	No (I29)	0,951149
HD 3000 (Decode)	Baseline	No	2	3772	No	No (I29)	0,965919
HD 3000 (Encode)	Baseline	No	2	4000	No	No (I29)	0,959298
HD 3000 (Full)	Baseline	No	2	4000	No	No (I29)	0,952697
HD 3000 (Decode)	Baseline	No	2	2743	No	No (I29)	0,951192
HD 3000 (Encode)	Baseline	No	2	3000	No	No (I29)	0,947684
HD 3000 (Full)	Baseline	No	2	3000	No	No (I80)	0,942121

It was a nice surprise to see that the AMD encoder didn't crash here and we were able to judge how good it is. If we take the averages for the whole of the extract, the classification seems to be as follows: CPU encoder, HD 3000 encoder, Radeon encoder, GeForce encoder.

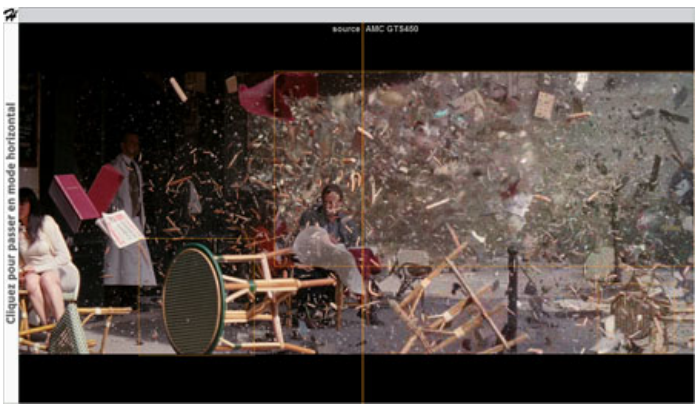
Let's see how the encoders do with the multiple scene changes in our extract:

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

The 'Full' encode peaks for the HD 3000 are no surprise as the scene is affected by the same bug as in Avatar:

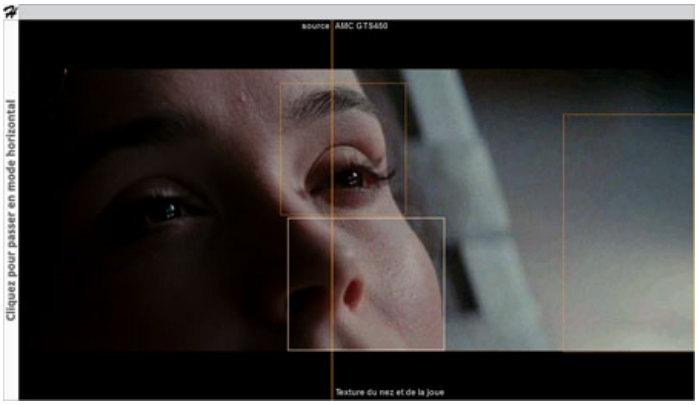


Otherwise, it's the explosions in the scene (it alternates between shots of characters and shots of massive explosions) that are making an impact between frames 250 and 350. What effect does this have visually?



Click here to display the frame comparison in a new tab.

Whatever the version, the results aren't good. The Radeon encoder gives a particularly blurred result and all the textures are lost. This is quite noticeable on the rattan table on the left. The NVIDIA encoder adds black marks to the red menu on the far left but conserves the textures on the rattan table better. Let's see if these encoders can make up any ground in a more static scene:



Click here to display the frame comparison in a new tab.

The CUDA encoder does a bit better on this frame with better conservation of details on the face in comparison to the others. In any case the grain on the right is lost whatever the encoder used.

K-On!! 720p

We finish with our animation, which theoretically should be the easiest scene for our encoders to deal with.

Cyberlink	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
CPU	Baseline	No	2	4104	No	No (I29)	0,985278
GeForce (Decode)	Baseline	No	2	4011	No	No (I29)	0,983085
GeForce (Encode)	Baseline	No	1	4000	No	No (I29)	0,578129
GeForce (Full)	Baseline	No	1	4000	No	No (I29)	0,576462
Radeon (Decode)	Baseline	No	2	4008	No	No (I29)	0,955365
Radeon (Encode)							
Radeon (Full)							
HD 3000 (Decode)	Baseline	No	2	4011	No	No (I29)	0,983632
HD 3000 (Encode)	Baseline	No	2	4000	No	No (I29)	0,98317
HD 3000 (Full)	Baseline	No	2	4000	No	No (I29)	0,988158
HD 3000 (Decode)	Baseline	No	2	3002	No	No (I29)	0,981677
HD 3000 (Encode)	Baseline	No	2	3000	No	No (I29)	0,982046
HD 3000 (Full)	Baseline	No	2	3000	No	No (I80)	0,987234

The Radeon encoder crashed again here.

Cyberlink	Time
CPU	510
AMD HD 6970 decode	371
AMD HD 6970 encode	X
AMD HD 6970 full	X
AMD HD 6850 decode	437
AMD HD 6850 encode	X
AMD HD 6850 full	X
AMD HD 5750 decode	453
AMD HD 5750 encode	X
AMD HD 5750 full	X
NV GTX 570 decode	417
NV GTX 570 encode	357
NV GTX 570 full	408
NV GTX 460 decode	417
NV GTX 460 encode	358
NV GTX 460 full	409
NV GTS 450 decode	417
NV GTS 450 encode	359
NV GTS 450 full	408
HD 3000 decode/qual	377
HD 3000 encode/qual	371
HD 3000 full/qual	357
HD 3000 decode/fast	264
HD 3000 encode/fast	363
HD 3000 full/fast	311

Again the GPU decode slows down the GPU encode.

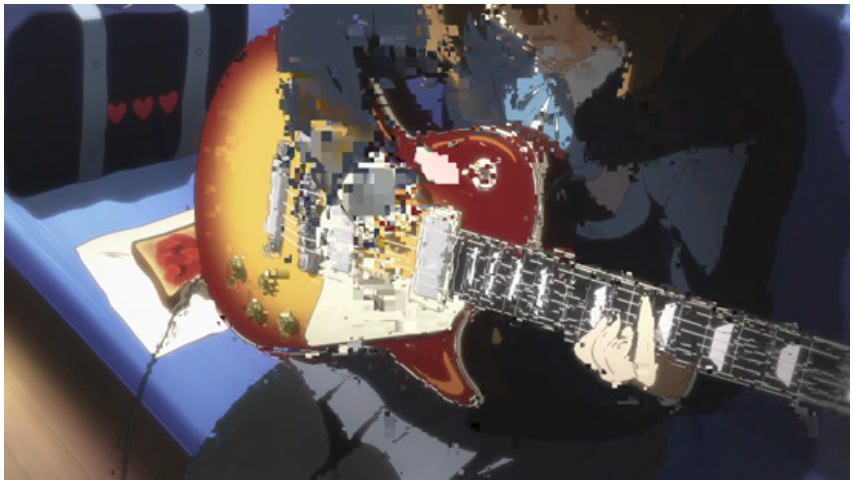
Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

Let's look at the problems on a case by case basis. First of all, the HD 3000 encoder continues its love affair with black squares:

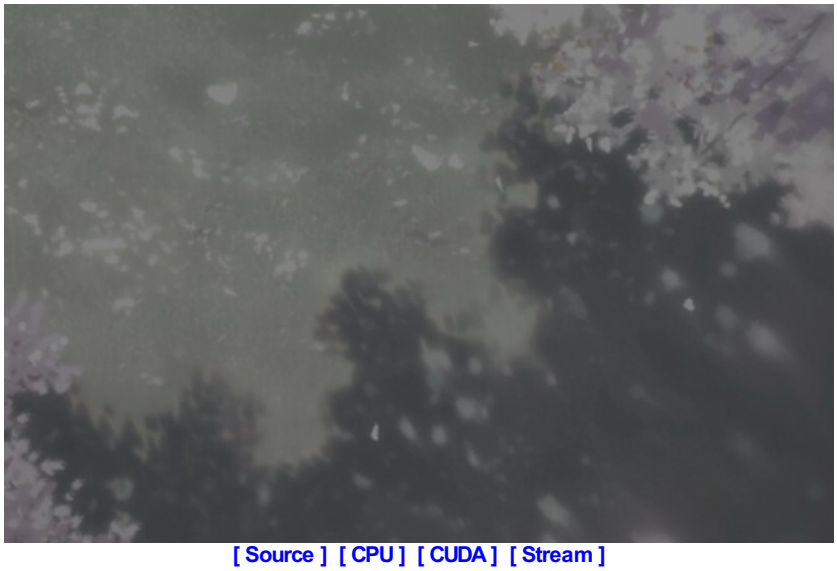
Next, a variable frame rate is used in the CUDA encoding, which puts frame comparison out of synch and renders it useless (via MediaInfo):

Original frame rate: 23.976 fps
Minimum frame rate: 0.237 fps
Maximum frame rate: 23.981 fps

The Radeon encoder doesn't work, but the decoder doesn't really work either as it produces this type of artefact (ironically more pronounced than with the 6970!):

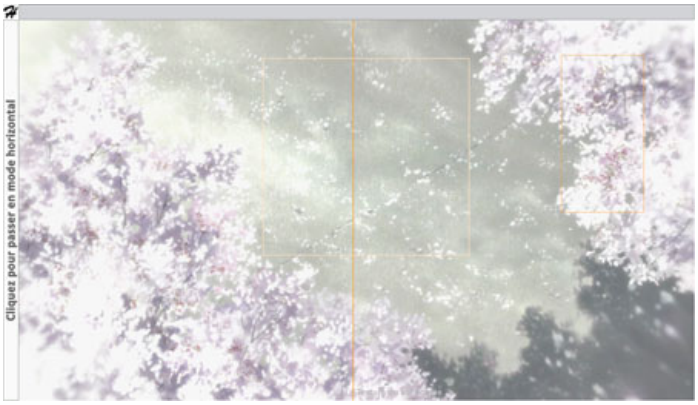


This is already pretty damning, but there's more. Whatever the encoder used, the violent deterioration shows up on the fades to black:



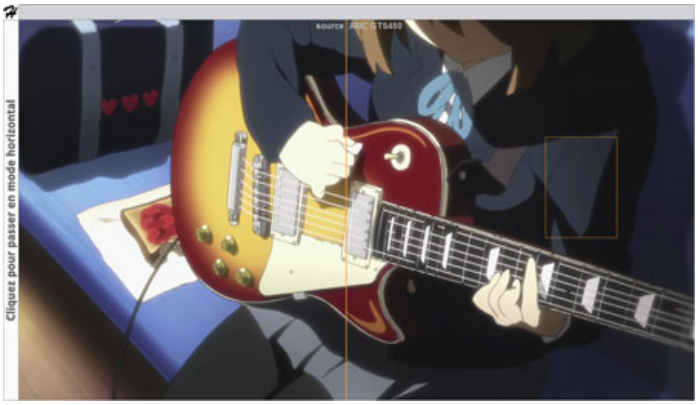
[Source] [CPU] [CUDA] [Stream]
Hold the mouse over/click on the links to display the corresponding frame.

Nevertheless, let's take a look at the visuals…



Click here to display the frame comparison in a new tab

You can see the Radeon HD 6970 artefacts in this scene. Enough said.



Click here to display the frame comparison in a new tab.

Just look at the results obtained following the Radeon 'decodes'. MediaEspresso does not do well with animation.

Page 14

Cyberlink MediaEspresso, Avatar 1080p (3/4)

Cyberlink MediaEspresso, Avatar 1080p

Will the Cyberlink software do any better at 1080p? To recap, we use a bitrate of 10 Mbit/s on the following extracts!

Cyberlink	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
CPU	Baseline	No	2	9822	No	No (I29)	0,680612
GeForce (Decode)	Baseline	No	2	9823	No	No (I29)	0,681213
GeForce (Encode)	Baseline	No	1	10000	No	No (I29)	0,962523
GeForce (Full)	Baseline	No	1	10000	No	No (I29)	0,962171
Radeon (Decode)	Baseline	No	2	9821	No	No (I29)	0,680608
Radeon (Encode)	Baseline	No	2	9946	No	No (I29)	0,76761
Radeon (Full)	Baseline	No	2	9947	No	No (I29)	0,767629
HD 3000 (Decode)	Baseline	No	2	9821	No	No (I29)	0,682304
HD 3000 (Encode)	Baseline	No	2	10000	No	No (I29)	0,683532
HD 3000 (Full)	Baseline	No	2	10000	No	No (I29)	0,963628
HD 3000 (Decode)	Baseline	No	2	8719	No	No (I29)	0,680609
HD 3000 (Encode)	Baseline	No	2	8000	No	No (I29)	0,681222
HD 3000 (Full)	Baseline	No	2	8000	No	No (I80)	0,967164

As at 720p, the GeForce and MediaSDK encoders have a constant bitrate. The Radeon encoder worked and the SSIM scores lead us to think that the frames are out of synch.

To be precise, the discrepancies are +3 frames in CPU mode, decode and HD 3000 encode and +1 frame in Radeon encode/full mode.

Cyberlink	Time	cpu%	Watts
CPU	458	100	109
AMD HD 6970 decode	407	98	183
AMD HD 6970 encode	308	30	106
AMD HD 6970 full	294	13	140
AMD HD 6850 decode	408	98	157
AMD HD 6850 encode	316	31	111
AMD HD 6850 full	307	13	114
AMD HD 5750 decode	406	98	136
AMD HD 5750 encode	313	27	102
AMD HD 5750 full	295	13	96
NV GTX 570 decode	407	97	185
NV GTX 570 encode	229	42	172
NV GTX 570 full	207	27	166
NV GTX 460 decode	407	97	144
NV GTX 460 encode	235	41	164
NV GTX 460 full	207	26	160
NV GTS 450 decode	406	98	163
NV GTS 450 encode	233	38	156
NV GTS 450 full	207	27	150
HD 3000 decode/qual	414	98	110
HD 3000 encode/qual	223	27	89
HD 3000 full/qual	194	11	70
HD 3000 decode/fast	230	83	108
HD 3000 encode/fast	214	27	90
HD 3000 full/fast	145	12	70

Note for once, GPU decoding doesn't slow down the encoding. We are giving you the links to our graphs below, in the interests of thoroughness, but the fact that the frames are out of synch makes the results impossible to compare:

Click here to see the SSIM graph for this scene.

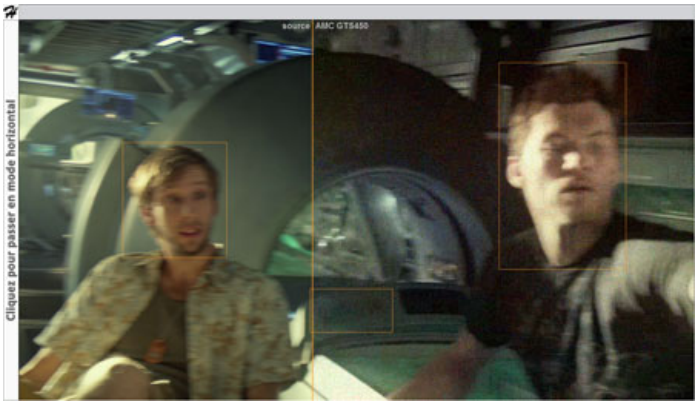
Click here to see the PSNR graph for this scene.

We compensated for the discrepancies manually to produce the visual comparison of the frames. Let's see what this gives us!

Click here to display the frame comparison in a new tab.

The results are rather blurred all in all. The CPU encoder version gives the best sharpness.

If you wish to see the graphs of our motion blur scene in spite of the frame discrepancies, click here for the SSIM readings and here for the PSNR. Looking at the visuals:



Click here to display the frame comparison in a new tab.

The HD 3000 version struggles, but all the versions give a measured result. The pure CPU encoding still gives the best result.

Page 15

Cyberlink MediaEspresso, Inception/K-On!! 1080p(4/4)

Cyberlink MediaEspresso, Inception 1080p

The Blu-ray of Inception was encoded with VC1. MediaEspresso won't open .m2ts files using this video codec.

K-On !! 1080p

Cyberlink	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
CPU	Baseline	No	2		No	No (I29)	0,938892
GeForce (Decode)	Baseline	No	2	10000	No	No (I29)	0,938713
GeForce (Encode)	Baseline	No	1	10000	No	No (I29)	0,991201
GeForce (Full)	Baseline	No	1	10000	No	No (I29)	0,990603
Radeon (Decode)	Baseline	No	2	10000	No	No (I29)	0,938889
Radeon (Encode)	Baseline	No	2	7747	No	No (I29)	0,938578
Radeon (Full)	Baseline	No	2	7767	No	No (I29)	0,938616
HD 3000 (Decode)	Baseline	No	2	10000	No	No (I29)	0,938242
HD 3000 (Encode)	Baseline	No	2	10000	No	No (I29)	0,938983
HD 3000 (Full)	Baseline	No	2	10000	No	No (I29)	0,959753
HD 3000 (Decode)	Baseline	No	2	8006	No	No (I29)	0,938888
HD 3000 (Encode)	Baseline	No	2	8000	No	No (I29)	0,93897
HD 3000 (Full)	Baseline	No	2	8000	No	No (I80)	0,9594

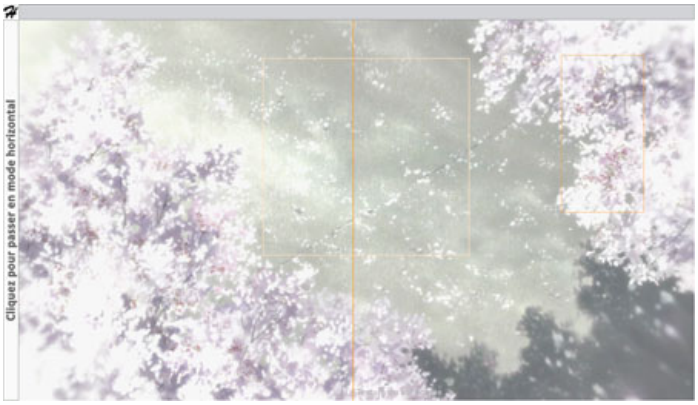
Can the crushing domination of the GeForces in the encode for K-On !! be verified in practice?

Cyberlink	Time
CPU	983
AMD HD 6970 decode	834
AMD HD 6970 encode	614
AMD HD 6970 full	754
AMD HD 6850 decode	835
AMD HD 6850 encode	591
AMD HD 6850 full	763
AMD HD 5750 decode	839
AMD HD 5750 encode	586
AMD HD 5750 full	788
NV GTX 570 decode	837
NV GTX 570 encode	620
NV GTX 570 full	668
NV GTX 460 decode	838
NV GTX 460 encode	625
NV GTX 460 full	669
NV GTS 450 decode	838
NV GTS 450 encode	616
NV GTS 450 full	668
HD 3000 decode/qual	854
HD 3000 encode/qual	597
HD 3000 full/qual	500
HD 3000 decode/fast	566
HD 3000 encode/fast	596
HD 3000 full/fast	381

GPU decoding slows the GPU encoding down once again…

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

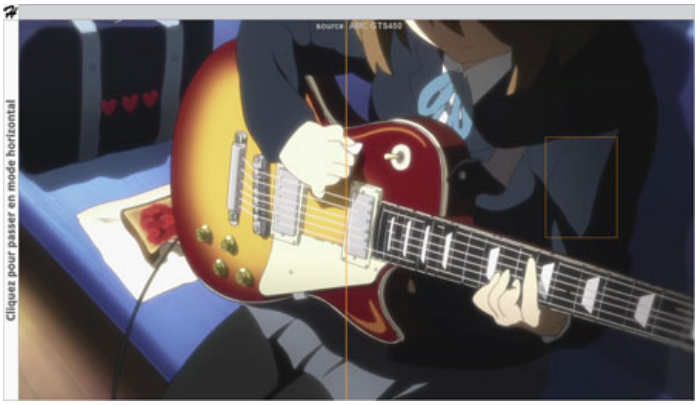
Once again, variable frame rates spoil any objective comparison. To illustrate what impact the fact that the frames are out of synch has, we haven't, for once, compensated for the differences so as to show you what our frame comparison software shows:



Click here to display the frame comparison in a new tab.

Here the differences due to frame discrepancy are minimal, with just a few flowers fluttering about differently.

The GeForce and CPU encoders are neck and neck in terms of conservation of details, with the Radeon version blurred and the black squares on the MediaSDK version being replaced by white squares!



Click here to display the frame comparison in a new tab.

Here the results are almost identical. Little does it matter however as the problems with the faded blacks that we noted at 720p are still there and the encodes are still unusable.

Summary

In only offering baseline profile H.264 encoding, MediaEspresso was already at a disadvantage in comparison to the competition software. The fact that it uses constant bitrates and variable frame rates doesn't help matters either, as the opposite (constant frame rate and variable bitrate!) is advisable when doing anything other than streaming. The absence of dynamic GOP tops things off by introducing constant jumps in quality whenever there are scene changes, as we showed previously:

Hold the mouse over the image to view the next I-frame.

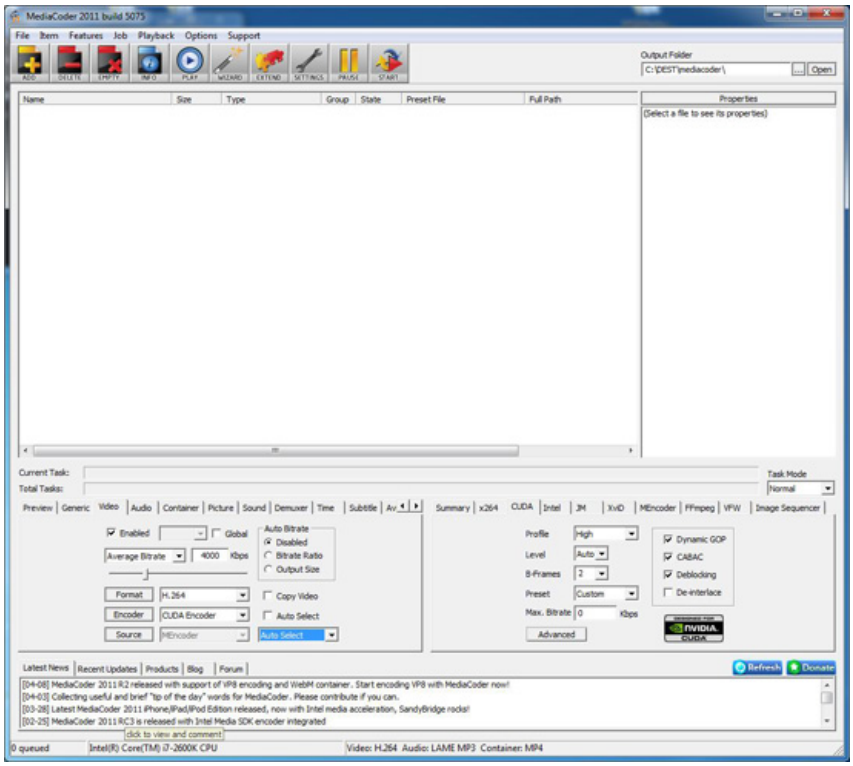
Cyberlink MediaEspresso CPU mode, Avatar, Fox Pathé Europa

Cyberlink could however easily correct most of these problems by configuring its encoders differently. Perhaps in a future version!

MediaCoder, Avatar 720p (1/4)

MediaCoder, Avatar 720p (1/4)

On offer from an independent developer, MediaCoder is a sort of ultimate video transcoding toolkit. Its author includes a multitude of encoding software in one package, including the official NVIDIA (CUDA) and Intel (MediaSDK) encoders. In practice MediaCoder comes with OpenCandy (adware) and, each time you start it up, the software opens your browser on a web page filled with ads (by launching the software minimised to the system tray!). On top of all this MediaCoder is listed in the ffmpeg hall of shame for violations of the GPL license.



Not really the sort of software you'd recommend on paper except that it does seem to have a couple of advantages: multithreaded CPU decoding and the ability to configure the NVIDIA/Intel encoders quite precisely.

Avatar 720p

MediaCoder	Profile	CABAC	Ref Frames	Bitrate	B Frames	Dynamic GOP
GeForce	Baseline	No	1	4075	No	No (I15)
GeForce	Main	Yes	2	4068	Yes	No (I15, 1P1B)
GeForce	High	Yes	2	4067	Yes	No (I15, 1P1B)
Intel	Baseline	No	2	3810	No	No (I256)
Intel	Main	Yes	2	3854	Yes	No (I256, 1P2B)
Intel	High	Yes	2	3833	Yes	No (I256, 1P2B)

MediaCoder	%I	%P	%B	SSIM	PSNR
GeForce	6,67	93,33	-	0,732094	22,95439
GeForce	6,67	46,66	46,66	0,73532	23,03147
GeForce	6,67	46,66	46,66	0,733901	23,02577
Intel	0,39	99,61	-	0,668427	20,17013
Intel	0,39	33,20	66,40	0,669297	20,19163
Intel	0,39	33,20	66,40	0,666764	20,18587

In theory MediaCoder allows you to turn on dynamic GOP for the GeForce encoder as well as giving you the choice between the three main H.264 profiles (baseline, main, high). Support for CABAC and B-frames is included but there's no dynamic GOP. The SSIM/PSNR scores are very low and for once, variable frame rate can't be blamed.

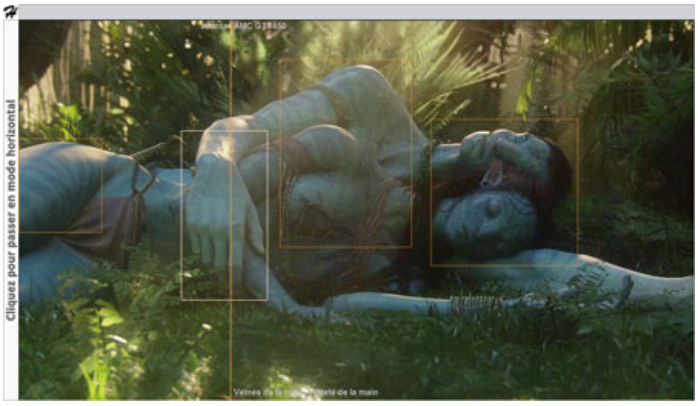
The version of MediaCoder that we used also tends to take out the first few frames at the beginning of the scene. In the case of Avatar, six frames have been taken out at the beginning on the NVIDIA encodes and four on the Intel encodes.

MediaCoder	Time
NV GTX 570 Baseline	92
NV GTX 570 Main	99
NV GTX 570 High	100
NV GTX 460 Baseline	89
NV GTX 460 Main	99
NV GTX 460 High	100
NV GTS 450 Baseline	91
NV GTS 450 Main	99
NV GTS 450 High	100
HD 3000 Quality/Baseline/1	135
HD 3000 Quality/Main/1	132
HD 3000 Quality/High/1	135
HD 3000 Quality/High/8	145

When it comes to encoding time, MediaCoder is unbeatable thanks to the use of a multithreaded CPU decoder in parallel with GPU encoding.

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

As the frames are out of synch, let's move on to the visual comparisons:

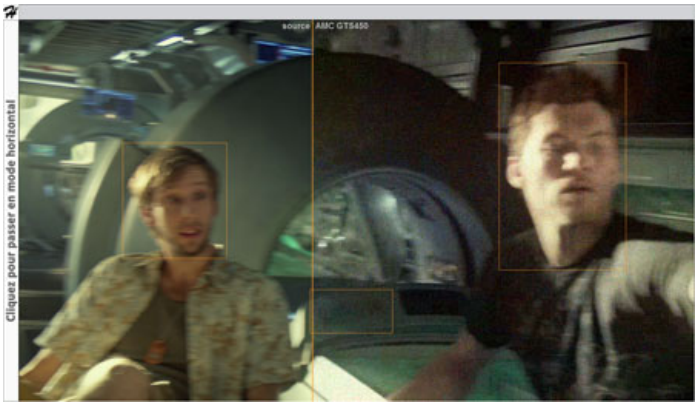


Click here to display the frame comparison in a new tab.

The results aren't necessarily that bad when you compare the higher encoding settings. Most of the textures disappear in both cases but the Intel encoder remains more blurred on faces than the rest of the encoders tested. As this tendency has now come up in three different applications, it looks as if it must be a fault of the Intel MediaSDK encoder.

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

The second graph isn't of much interest because of the frame discrepancies, so we'll move on to the images resulting from motion blur:



Click here to display the frame comparison in a new tab.

In this scene the Intel encoder retains a small advantage with respect to the character on the left.

Page 17

MediaCoder, Inception/K-On!! 720p (2/4)

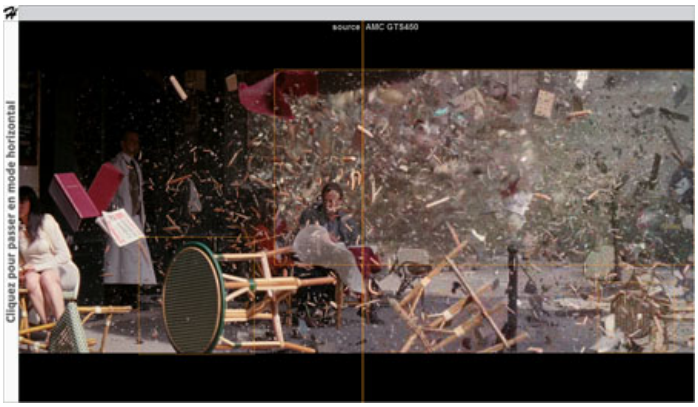
MediaCoder, Inception 720p

MediaCoder	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
GeForce	Baseline	No	1	4092	No	No (I15)	0,863072
GeForce	Main	Yes	2	4066	Yes	No (I15, 1P1B)	0,869405
GeForce	High	Yes	2	4067	Yes	No (I15, 1P1B)	0,869323
Intel	Baseline	No	2	3523	No	No (I256)	0,828705
Intel	Main	Yes	2	3556	Yes	No (I256, 1P2B)	0,831682
Intel	High	Yes	2	3672	Yes	No (I256, 1P2B)	0,831181

Once again there's a discrepancy of 6 frames on the GeForce version and 4 frames on the HD 3000 version.

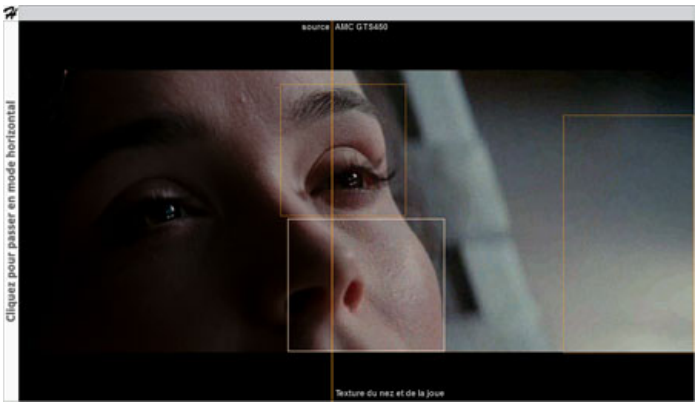
Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

We'll leave the graphs to one side and compare the images:



Click here to display the frame comparison in a new tab.

By comparing the baseline and high GeForce modes, you quickly see how Cyberlink would benefit from better configuration of its encoders! The Intel encoder is a bit more blurred and does better on complex textures such as the table on the left. The GeForce version tends to retain more detail but create artefacts.



Click here to display the frame comparison in a new tab.

In our static scene, the difference in sharpness between the baseline and high modes is high in both cases, proving once again that it is best to use the highest possible profile. The GeForce encoder is still a little sharper.

K-On!! 720p

MediaCoder	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
GeForce	Baseline	No	1	4007	No	No (I15)	0,95453
GeForce	Main	Yes	2	4006	Yes	No (I15, 1P1B)	0,954954
GeForce	High	Yes	2	4006	Yes	No (I15, 1P1B)	0,954794
Intel	Baseline	No	2	3940	No	No (I256)	0,930727
Intel	Main	Yes	2	3867	Yes	No (I256, 1P2B)	0,930941
Intel	High	Yes	2	3879	Yes	No (I256, 1P2B)	0,930335

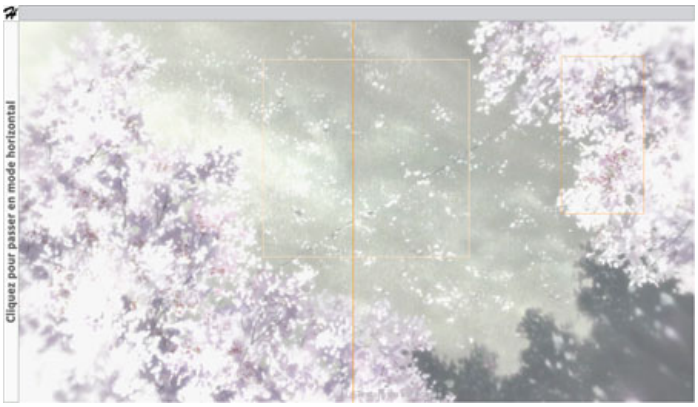
The frame discrepancies are smaller here, just 1 or 2 frames, which is the reason for the higher scores.

MediaCoder	Time
NV GTX 570 Baseline	229
NV GTX 570 Main	236
NV GTX 570 High	252
NV GTX 460 Baseline	236
NV GTX 460 Main	238
NV GTX 460 High	256
NV GTS 450 Baseline	246
NV GTS 450 Main	251
NV GTS 450 High	255
HD 3000 Quality/Baseline/1	300
HD 3000 Quality/Main/1	316
HD 3000 Quality/High/1	324
HD 3000 Quality/High/8	331

MediaCoder is still the fastest of the encoders in our panel.

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

It's very easy to see each change in scene or each movement by looking at the peaks and troughs! Let's move on to the much more useful image comparisons:

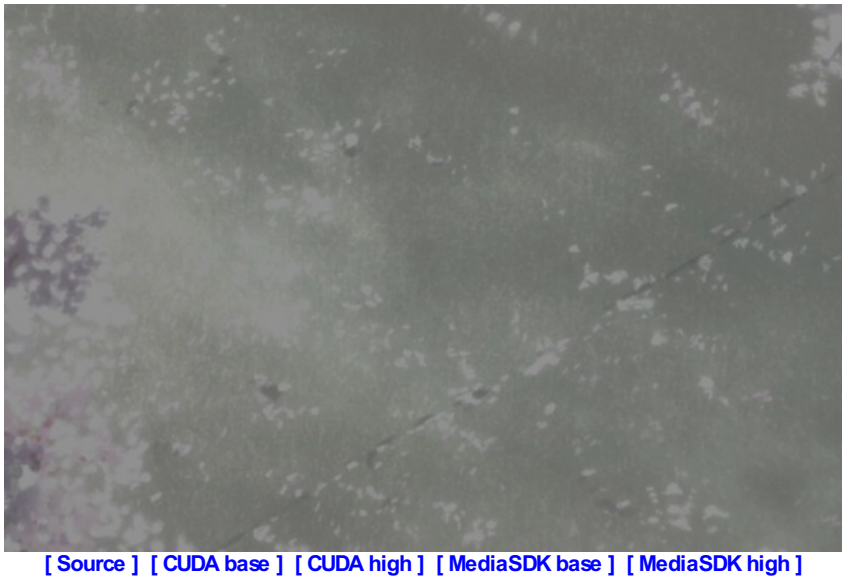


Click here to display the frame comparison in a new tab.

The CUDA encoder does well with this scene, whatever the software used, and the result is sharp here as of baseline mode. High gives very good results indeed. The HD 3000 version is a notch down here.

Click here to display the frame comparison in a new tab.

The differences are very minimal here and slight artefacts continue to appear in solid colours. Note that the problems in the fade to blacks that we observed with MediaEspresso persist here… but only in the base profiles:



[Source] [CUDA base] [CUDA high] [MediaSDK base] [MediaSDK high]
Hold the mouse over/click on the links to display the corresponding image.

Page 18

MediaCoder, Avatar 1080p (3/4)

MediaCoder, Avatar 1080p

The 1080p encoding was carried out at a bitrate of 10 Mbit/s.

MediaCoder	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
GeForce	Baseline	No	1	10200	No	No (I15)	0,964558
GeForce	Main	Yes	2	10100	Yes	No (I15, 1P1B)	0,966629
GeForce	High	Yes	2	10100	Yes	No (I15, 1P1B)	0,967854
Intel	Baseline	No	2	9520	No	No (I256)	0,96875
Intel	Main	Yes	2	9546	Yes	No (I256, 1P2B)	0,970067
Intel	High	Yes	2	9575	Yes	No (I256, 1P2B)	0,971124

The good news is that, while using an MKV file as the source caused frame discrepancy at 720p, using an m2ts file as the source means the frames are not out of synch! Phew!

MediaCoder	Time	cpu%	Watts
NV GTX 570 Baseline	198	93	202
NV GTX 570 Main	220	93	202
NV GTX 570 High	224	93	203
NV GTX 460 Baseline	203	93	190
NV GTX 460 Main	223	94	192
NV GTX 460 High	226	94	192
NV GTS 450 Baseline	198	93	183
NV GTS 450 Main	220	93	184
NV GTS 450 High	232	93	184
HD 3000 Quality/Baseline/1	211	66	106
HD 3000 Quality/Main/1	224	70	107
HD 3000 Quality/High/1	229	71	109
HD 3000 Quality/High/8	222	71	109

In confirmation of what we were saying earlier, using a CPU decoder in parallel with a GPU encoder makes MediaCoder extremely fast.

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

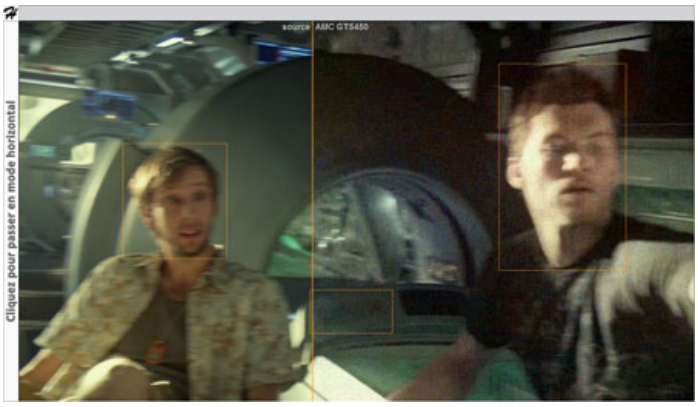
Intriguingly the quality of the Intel encoder on the tracking shot is slightly better in base mode than high. However the results remain very consistent throughout the extract. Let's check out the quality in practice after a change in scene.

Click here to display the frame comparison in a new tab.

Without dynamic GOP, the loss in sharpness is very significant on the Intel encoder, a little less on the CUDA encoder which gives a very decent result in high mode.

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

In our second scene the quality levels seem to be very comparable once again, with an advantage going to the Intel encoder. What about the visual image?



Click here to display the frame comparison in a new tab.

Intriguingly, a green square is at first very visible on the NVIDIA encoder results. Showing up most on the character on the left, squares also appear on the shaded areas of the middle/upper part of the pod. The Intel version is more blurred but doesn't suffer from these issues.

Page 19

MediaCoder, Inception/K-On!! 1080p(4/4)

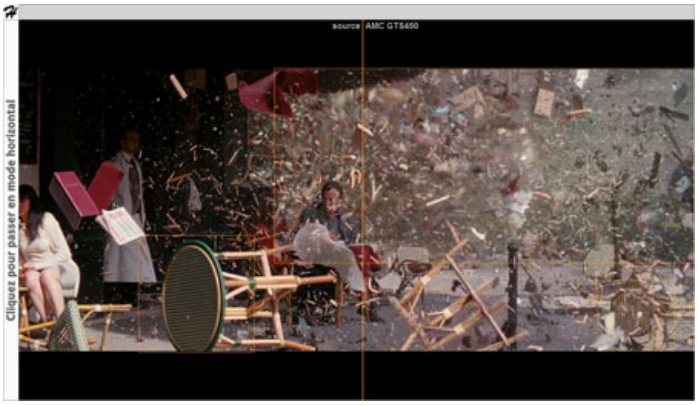
MediaCoder, Inception 1080p

MediaCoder	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
GeForce	Baseline	No	1	10200	No	No (I15)	0,657858
GeForce	Main	Yes	2	10100	Yes	No (I15, 1P1B)	0,656172
GeForce	High	Yes	2	10100	Yes	No (I15, 1P1B)	0,655235
Intel	Baseline	No	2	8222	No	No (I256)	0,659096
Intel	Main	Yes	2	8219	Yes	No (I256, 1P2B)	0,656972
Intel	High	Yes	2	8323	Yes	No (I256, 1P2B)	0,655506

MediaCoder redefines the notion of frame discrepancy here as 24 frames are missing from the beginning of the scene! A full second, no less…

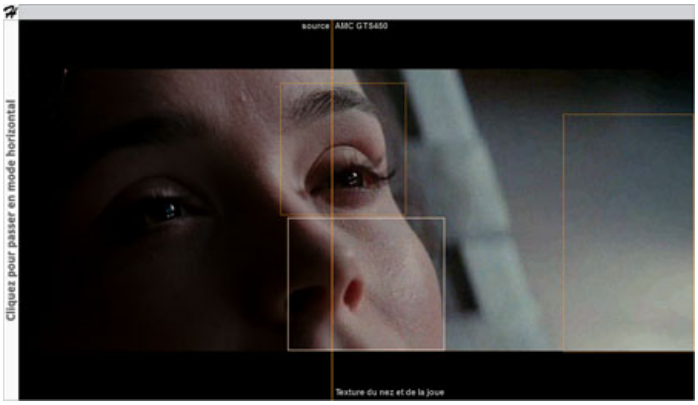
Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

The second delay here makes the graphs useless. Let's move on to the visual comparisons:



Click here to display the frame comparison in a new tab.

The advantage of the Intel encoder in terms of conservation of textures is particularly visible on the rattan table. On the other hand, the NVIDIA version creates colour nuances which don't exist in the source image and which translate into visual artefacts when the frames succeed one after the next.



Click here to display the frame comparison in a new tab.

If we forget the grain in the scene on the right, the NVIDIA encoder is once again sharper, when it comes to the face, than the Intel encoder.

K-On !! 1080p

So then, frame discrepancy or not? Listen to the drum roll!

MediaCoder	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
GeForce	Baseline	No	1	10000	No	No (I15)	0,990201
GeForce	Main	Yes	2	10000	Yes	No (I15, 1P1B)	0,990876
GeForce	High	Yes	2	10000	Yes	No (I15, 1P1B)	0,990666
Intel	Baseline	No	2	9683	No	No (I256)	0,990686
Intel	Main	Yes	2	9730	Yes	No (I256, 1P2B)	0,991458
Intel	High	Yes	2	9701	Yes	No (I256, 1P2B)	0,990603

The answer is… no. This time our frames are correctly aligned.

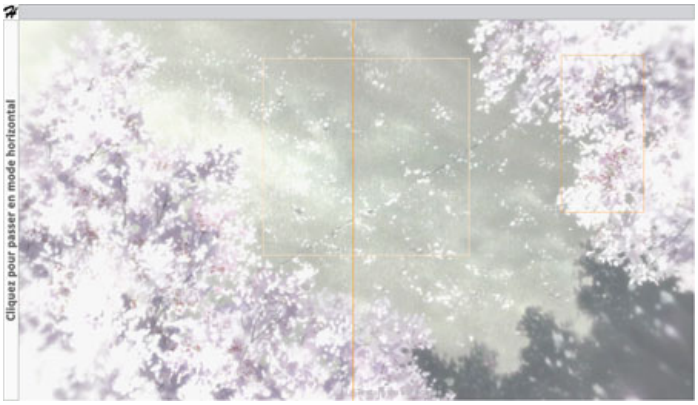
MediaCoder	Time
NV GTX 570 Baseline	541
NV GTX 570 Main	582
NV GTX 570 High	571
NV GTX 460 Baseline	529
NV GTX 460 Main	533
NV GTX 460 High	540
NV GTS 450 Baseline	545
NV GTS 450 Main	583
NV GTS 450 High	590
HD 3000 Quality/Baseline/1	620
HD 3000 Quality/Main/1	610
HD 3000 Quality/High/1	637
HD 3000 Quality/High/8	618

GPU decoding is again slower here! Let's look at the results:

Use an HTML5 compatible browser to view the graph!

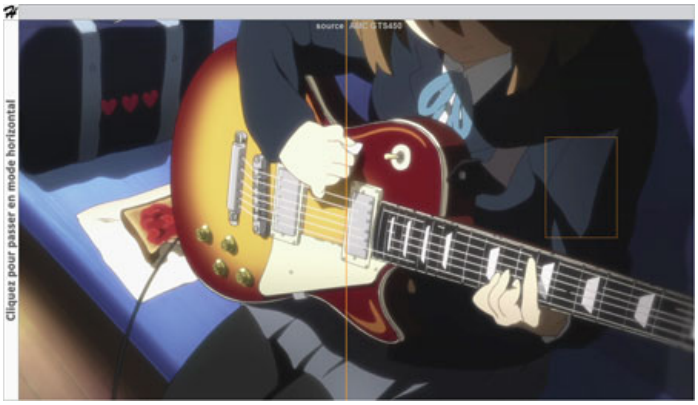
Click here to view the PSNR graph of this scene.

Quality is pretty homogenous throughout the scene, with just the baseline modes struggling a bit during the fade in/fade out.



Click here to display the frame comparison in a new tab.

Upwards of base mode, none of the encoders has any problem displaying a static image during several seconds.



Click here to display the frame comparison in a new tab.

There are very few problems on this second image.

Summary

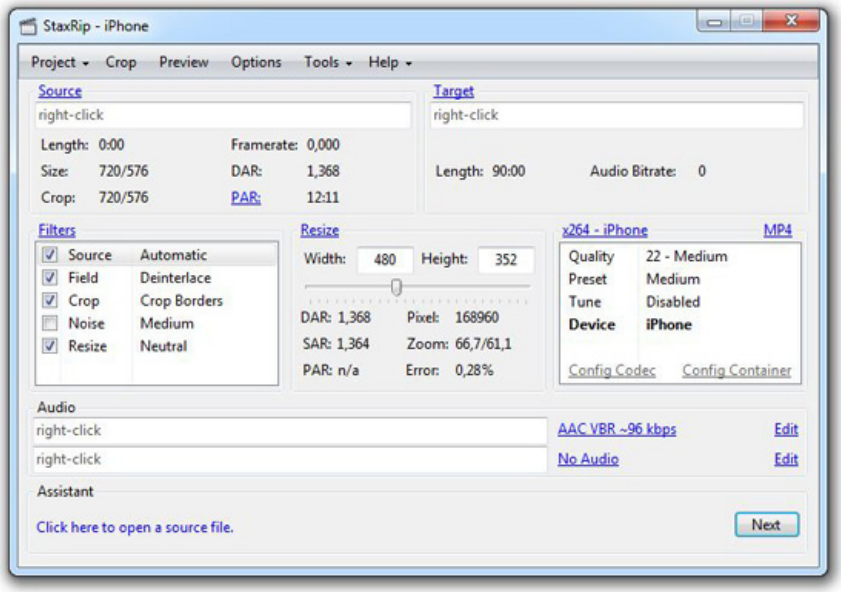
Overall, the results obtained with MediaCoder were good and being able to use the high H.264 profile gives a boost to quality, particularly in the CUDA version of the encoder which was held back by poor implementation in the Arcsoft application and by the baseline profile in Cyberlink. In spite of this, while the results are good, we do need to put them into context: the results are good for GPU encoding. However, missing frames, too much advertising and the unfriendly interface mean this doesn't come without a cost, even if the software is itself free.

Page 20

StaxRip/x264, Avatar 720p (1/4)

StaxRip/x264, Avatar 720p (1/4)

As we mentioned earlier, we wanted to compare H.264 encoder GPU-accelerated performance with what exists on the CPU side. While a great deal of commercial software exists, there's also an Open Source implementation of H.264 known as x264. A project started by some of the VLC authors, x264 is an open source implementation of H.264, just as XviD was in its time for Mpeg 4 part 2. x264 is generally considered to be the best in terms of quality, to the detriment of encoding speed of course.



In itself x264 handles only the encoding of raw frames to the H.264 format. In order to carry out transcoding, you need to use a frontend which includes the various other necessary bits and pieces (decoders, audio encoders and so on). We used StaxRip, which is once again an open source application. While it is a little more complex to get used to than Arcsoft or Cyberlink, its interface isn't too bad. It has the advantage of offering plenty of control in terms of x264 configuration, which is why we opted for it over other tools such as Handbrake which we recommend if you find StaxRip too hard to get a handle on.

We used version 1.1.7.0 of StaxRip, to which we added the latest build of x264 to date at the beginning of our tests, r1913.
In contrast to the other applications, x264 literally allows you to configure all the encoding parameters. This is very practical, though it can sometimes seem a bit intimidating:

cabac=1 / ref=16 / deblock=1:-1:-1 / analyse=0x3:0x133 / me=umh / subme=10 / psy=1 / psy_rd=1.00:0.15 / mixed_ref=1 / me_range=24 / chroma_me=1 / trellis=2 / 8x8dct=1 / cqm=0 / deadzone=21,11 / fast_pskip=1 / chroma_qp_offset=-3 / threads=12 / sliced_threads=0 / nr=0 / decimate=1 / interlaced=0 / constrained_intra=0 / bframes=8 / b_pyramid=2 / b_adapt=2 / b_bias=0 / direct=3 / weightb=1 / open_gop=0 / weightp=2 / keyint=250 / keyint_min=23 / scenecut=40 / intra_refresh=0 / rc_lookahead=60 / rc=2pass / mbtree=1 / bitrate=4029 / ratetol=1.0 / qcomp=0.60 / qpmin=0 / qpmax=69 / qpstep=4 / cplxblur=20.0 / qblur=0.5 / ip_ratio=1.40 / aq=1:1.00

In order to simplify the task a bit, the authors have also added a number of presets, which offer successively higher quality levels. For example, in the above encode, we've simply gone for 'veryslow' mode. There are 10 different modes, going from 'ultrafast' to 'placebo', an awfully slow mode on which the gains in comparison to 'veryslow' are barely visible.

We tested nine of the modes (all except 'placebo') both with two passes (the standard and recommended mode) and one pass to provide comparison with our other encoders. It's time to show you our findings!

Avatar 720p

StaxRip/x264	Profile	CABAC	Ref Frames	Bitrate	B Frames	Dynamic GOP
ultrafast (1 pass)	Baseline	No	1	4029	No	No (I250)
superfast (1 pass)	High	Yes	4	3831	Yes	Yes
veryfast (1 pass)	High	Yes	4	3801	Yes	Yes
faster (1 pass)	High	Yes	4	3813	Yes	Yes
fast (1 pass)	High	Yes	4	3818	Yes	Yes
medium (1 pass)	High	Yes	4	3813	Yes	Yes
slow (1 pass)	High	Yes	5	3818	Yes	Yes
slower (1 pass)	High	Yes	8	3815	Yes	Yes
veryslow (1 pass)	High	Yes	16	3814	Yes	Yes
ultrafast (2 pass)	Baseline	No	1	4029	No	No (I250)
superfast (2 pass)	High	Yes	4	4029	Yes	Yes
veryfast (2 pass)	High	Yes	4	4029	Yes	Yes
faster (2 pass)	High	Yes	4	4029	Yes	Yes
fast (2 pass)	High	Yes	4	4029	Yes	Yes
medium (2 pass)	High	Yes	4	4029	Yes	Yes
slow (2 pass)	High	Yes	5	4029	Yes	Yes
slower (2 pass)	High	Yes	8	4029	Yes	Yes
veryslow (2 pass)	High	Yes	16	4029	Yes	Yes

StaxRip/x264	%I	%P	%B	SSIM	PSNR
ultrafast (1 pass)	0,40	99,60	-	0,928665	35,05235
superfast (1 pass)	1,28	44,19	54,53	0,963517	37,33435
veryfast (1 pass)	1,51	51,78	46,71	0,966638	37,77035
faster (1 pass)	1,51	51,78	46,71	0,97001	38,44325
fast (1 pass)	1,51	51,78	46,71	0,969003	38,64822
medium (1 pass)	1,51	51,78	46,71	0,968857	38,69058
slow (1 pass)	1,10	29,49	69,41	0,96891	38,79686
slower (1 pass)	1,10	29,49	69,41	0,969431	38,92523
veryslow (1 pass)	0,84	26,38	72,78	0,969743	38,93384
ultrafast (2 pass)	0,40	99,60	-	0,932722	35,12283
superfast (2 pass)	1,28	44,19	54,53	0,965634	37,4711
veryfast (2 pass)	1,52	52,81	45,67	0,96921	37,80453
faster (2 pass)	1,52	52,81	45,67	0,97222	38,48787
fast (2 pass)	1,52	52,81	45,67	0,971226	38,69144
medium (2 pass)	1,52	52,81	45,67	0,971053	38,74487
slow (2 pass)	1,09	30,08	68,83	0,971032	38,84375
slower (2 pass)	1,09	30,08	68,83	0,97145	38,96787
veryslow (2 pass)	0,80	27,28	71,92	0,971786	38,98653

'Ultrafast' mode in tune film implies a baseline H.264 profile.

StaxRip/x264	Time	StaxRip/x264	Time
ultrafast (1 pass)	265	ultrafast (2 pass)	515
superfast (1 pass)	271	superfast (2 pass)	524
veryfast (1 pass)	287	veryfast (2 pass)	536
faster (1 pass)	330	faster (2 pass)	580
fast (1 pass)	389	fast (2 pass)	627
medium (1 pass)	434	medium (2 pass)	671
slow (1 pass)	644	slow (2 pass)	886
slower (1 pass)	1329	slower (2 pass)	1596
veryslow (1 pass)	2503	veryslow (2 pass)	3029

As you can see with the processing times, the addition of a second pass doesn't double encoding time. By definition, the first pass is rapid. We've pulled out the processing times for each pass for information:

	Total	passe 1	passe 2
ultrafast	515	250	253
superfast	524	255	257
veryfast	536	259	266
faster	580	266	302
fast	627	267	348
medium	671	274	385
slow	886	295	579
slower	1596	297	1287
veryslow	3029	619	2397

The 2p faster mode was carried out in real time on our Core i7.

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

As you can see, the 'ultrafast' preset stands out from the rest. This is hardly surprising as it's the only preset to use the baseline profile. x264 is indeed configured to give the fastest possible result, to compete with the GPU encoders. In practice this mode should be avoided as the results are awful to say the least, as we'll see. Note that all the other modes use a high profile (with CABAC, B-frames, dynamic GOP and so on).

If you compare one preset in single and double pass (2p) mode, you can straightaway see which is best in quality terms. How does all this translate visually?

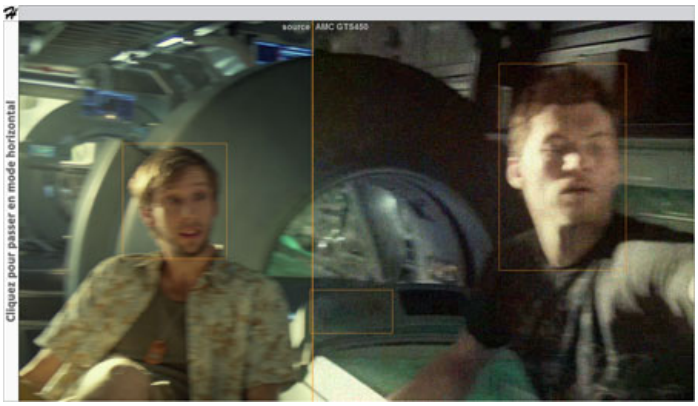
Click here to display the frame comparison in a new tab.

While 'ultrafast' mode is truly bad for your eyes, quality rises quickly. In 'faster' mode, the quality is already higher than the best of what we've seen up to now. What is much more interesting however is that with two passes there's a significant improvement in quality, with veryfast 2p already significantly better.

Now let's see what we get with motion blur:

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

Note that x264 encoding isn't optimised for SSIM measurements, but rather for visual quality according to psychovisual optimisations. In spite of everything, as of 'veryfast' (1 pass), quality is up on everything we've seen up until now. What about the visual image?



Click here to display the frame comparison in a new tab.

As of 'superfast' mode, the visual quality is higher, especially when you look at the grille in the middle. Once again, the 2p modes give significantly more sharpness.

Page 21

StaxRip/x264, Inception/K-On!! 720p (2/4)

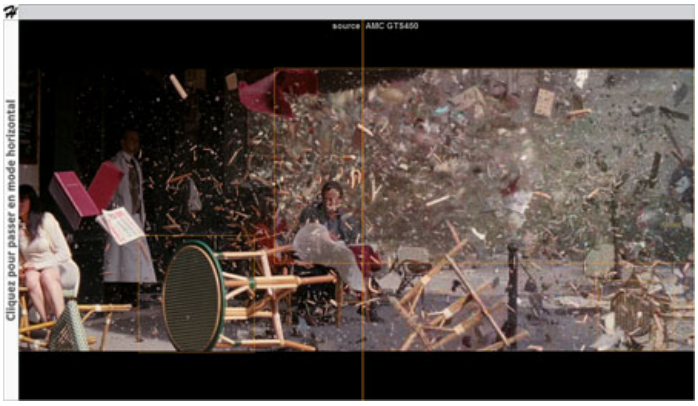
StaxRip/x264, Inception 720p

StaxRip/x264	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
ultrafast (1 pass)	Baseline	No	1	3667	No	No (I250)	0,934562
superfast (1 pass)	High	Yes	4	3618	Yes	Yes	0,968027
veryfast (1 pass)	High	Yes	4	3440	Yes	Yes	0,970862
faster (1 pass)	High	Yes	4	3462	Yes	Yes	0,973986
fast (1 pass)	High	Yes	4	3469	Yes	Yes	0,973751
medium (1 pass)	High	Yes	4	3458	Yes	Yes	0,973457
slow (1 pass)	High	Yes	5	3486	Yes	Yes	0,974147
slower (1 pass)	High	Yes	8	3478	Yes	Yes	0,974689
veryslow (1 pass)	High	Yes	16	3442	Yes	Yes	0,974753
ultrafast (2 pass)	Baseline	No	1	4061	No	No (I250)	0,946787
superfast (2 pass)	High	Yes	4	3861	Yes	Yes	0,972768
veryfast (2 pass)	High	Yes	4	3842	Yes	Yes	0,97688
faster (2 pass)	High	Yes	4	4061	Yes	Yes	0,979433
fast (2 pass)	High	Yes	4	4061	Yes	Yes	-
medium (2 pass)	High	Yes	4	4061	Yes	Yes	0,979097
slow (2 pass)	High	Yes	5	4061	Yes	Yes	0,979249
slower (2 pass)	High	Yes	8	4061	Yes	Yes	0,979631
veryslow (2 pass)	High	Yes	16	3867	Yes	Yes	0,979969

Let's move on to our choice morsel, our explosion scenes!

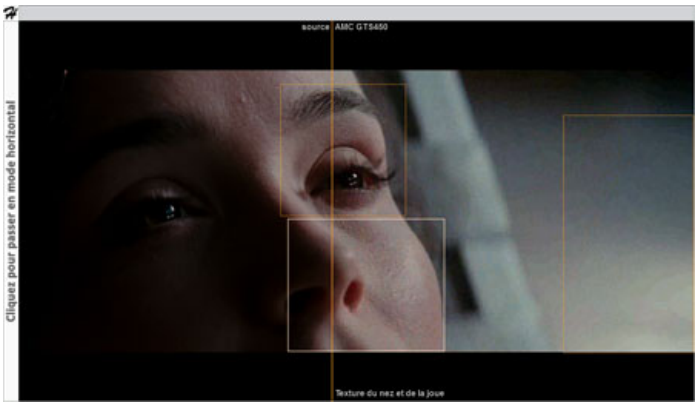
Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

Once again, it is particularly interesting to compare the scenes with character's faces with the explosion scenes. In single pass mode, you can see very clearly that the visual image is a little down in terms of quality. In double pass mode however, the encoder recognises the difficulty of the scene and budgets a higher bitrate for these scenes. Visually, the difference is notable:



Click here to display the frame comparison in a new tab.

Look how easily a mode such as 'veryfast' 2p outdoes 'veryslow' 1p, which is nevertheless 3.6x slower. As of 'veryfast' mode, double pass quality is almost perfect!



Click here to display the frame comparison in a new tab.

The results in single pass mode aren't that great. Note that even 'veryslow' 2p doesn't conserve the original grain of our scene on the right. The bitrate simply doesn't allow this.

K-On !! 720p

StaxRip/x264	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
ultrafast (1 pass)	Main	No	4	4037	No	No (I250)	0,991695
superfast (1 pass)	High	Yes	4	4037	Yes	Yes	0,992418
veryfast (1 pass)	High	Yes	4	4037	Yes	Yes	0,993219
faster (1 pass)	High	Yes	4	4037	Yes	Yes	0,993878
fast (1 pass)	High	Yes	4	4037	Yes	Yes	0,993819
medium (1 pass)	High	Yes	6	4037	Yes	Yes	0,99386
slow (1 pass)	High	Yes	10	4037	Yes	Yes	0,994041
slower (1 pass)	High	Yes	16	4037	Yes	Yes	0,994284
veryslow (1 pass)	High	Yes	16	4037	Yes	Yes	0,994394
ultrafast (2 pass)	Main	No	4	4037	No	No (I250)	0,992671
superfast (2 pass)	High	Yes	4	4037	Yes	Yes	0,992664
veryfast (2 pass)	High	Yes	4	4037	Yes	Yes	0,993313
faster (2 pass)	High	Yes	4	4037	Yes	Yes	0,993964
fast (2 pass)	High	Yes	4	4037	Yes	Yes	0,993885
medium (2 pass)	High	Yes	6	4037	Yes	Yes	0,993907
slow (2 pass)	High	Yes	10	4037	Yes	Yes	0,994077
slower (2 pass)	High	Yes	16	4037	Yes	Yes	0,994316
veryslow (2 pass)	High	Yes	16	4037	Yes	Yes	0,994421

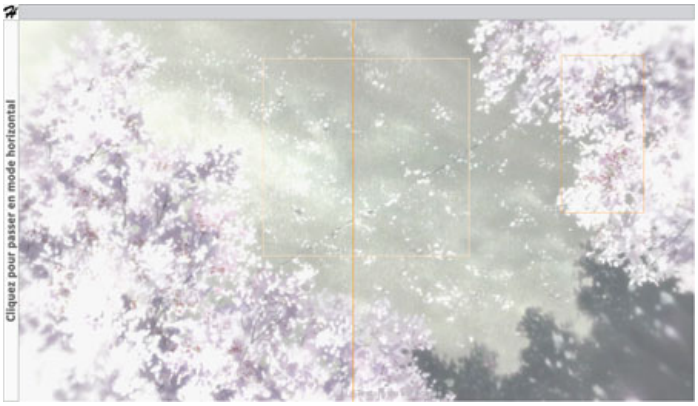
For K-On!!, we went for the tune animation mode in x264.

StaxRip/x264	Time	StaxRip/x264	Time
ultrafast (1 pass)	682	ultrafast (2 pass)	1324
superfast (1 pass)	685	superfast (2 pass)	1339
veryfast (1 pass)	693	veryfast (2 pass)	1344
faster (1 pass)	779	faster (2 pass)	1417
fast (1 pass)	870	fast (2 pass)	1502
medium (1 pass)	1006	medium (2 pass)	1637
slow (1 pass)	1478	slow (2 pass)	2092
slower (1 pass)	3275	slower (2 pass)	3913
veryslow (1 pass)	4810	veryslow (2 pass)	5692

The ‘faster’ 2p mode is encoded in real time on our processor.

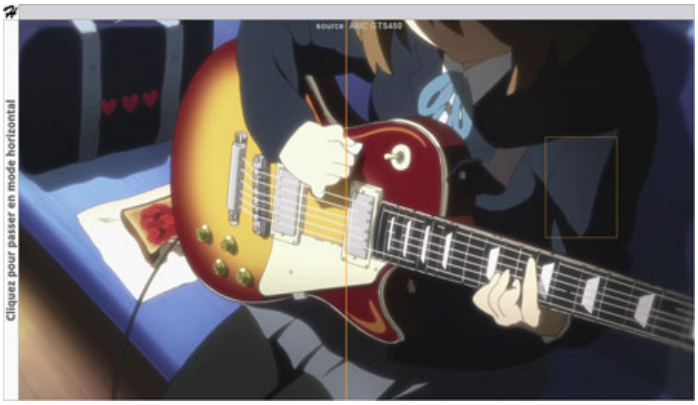
Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

The scores are very high here, even in ‘ultrafast’ mode. The advantage given by 2p mode isn’t as clear here, at least not in this sequence. And how does it look?



Click here to display the frame comparison in a new tab.

As of 'superfast’ the quality is already very close to our source image! ‘Ultrafast’ is sharper than most GPU baseline encodes.



Click here to display the frame comparison in a new tab.

In 'ultrafast' mode an artefact appears in the solid blue area isolated on screen. The other modes are perfect visually speaking.

Page 22

StaxRip/x264, Avatar 1080p (3/4)

StaxRip/x264, Avatar 1080p

Moving on to 1080p and a bitrate of 10 Mbit/s…

StaxRip/x264	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
ultrafast (1 pass)	Baseline	No	1	10100	No	No (I250)	0,94997
superfast (1 pass)	High	Yes	4	9568	Yes	Yes	0,969402
veryfast (1 pass)	High	Yes	4	9491	Yes	Yes	0,971692
faster (1 pass)	High	Yes	4	9518	Yes	Yes	0,974361
fast (1 pass)	High	Yes	4	9525	Yes	Yes	0,972979
medium (1 pass)	High	Yes	4	9505	Yes	Yes	0,972735
slow (1 pass)	High	Yes	5	9492	Yes	Yes	0,972989
slower (1 pass)	High	Yes	8	9477	Yes	Yes	0,973309
veryslow (1 pass)	High	Yes	16	9468	Yes	Yes	0,973428
ultrafast (2 pass)	Baseline	No	1	10100	No	No (I250)	0,952601
superfast (2 pass)	High	Yes	4	10100	Yes	Yes	0,970716
veryfast (2 pass)	High	Yes	4	10100	Yes	Yes	0,973222
faster (2 pass)	High	Yes	4	10100	Yes	Yes	0,975686
fast (2 pass)	High	Yes	4	10100	Yes	Yes	0,974216
medium (2 pass)	High	Yes	4	10100	Yes	Yes	0,973901
slow (2 pass)	High	Yes	5	10100	Yes	Yes	0,97421
slower (2 pass)	High	Yes	8	10100	Yes	Yes	0,974544
veryslow (2 pass)	High	Yes	16	10100	Yes	Yes	0,974743

StaxRip/x264	Time	cpu%	Watts	Time	cpu%	Watts
ultrafast (1 pass)	166	74	99			
superfast (1 pass)	223	83	103			
veryfast (1 pass)	398	72	97			
faster (1 pass)	567	99	106			
fast (1 pass)	788	98	106			
medium (1 pass)	945	98	106			
slow (1 pass)	1595	98	106			
slower (1 pass)	3343	98	105			
veryslow (1 pass)	6327	97	103			
ultrafast (2 pass)	145	74	99	138	69	96
superfast (2 pass)	189	85	104	175	94	106
veryfast (2 pass)	314	52	90	242	99	106
faster (2 pass)	326	51	89	459	100	106
fast (2 pass)	345	48	85	696	100	107
medium (2 pass)	355	48	85	846	100	107
slow (2 pass)	633	34	79	1451	96	104
slower (2 pass)	643	32	72	3233	99	105
veryslow (2 pass)	1533	23	68	6046	99	105

If you've got sharp eyes you'll have noticed that the 'ultrafast' and 'superfast' modes are faster here at 1080p than at 720p! Our 720p source file has an absurdly high bitrate (for the quantity of pixels) and the encoding was slowed down… by the decoding. The StaxRip decoder isn't multithreaded.

Use an HTML5 compatible browser to view the graph!

Click here to view the PSNR graph of this scene.

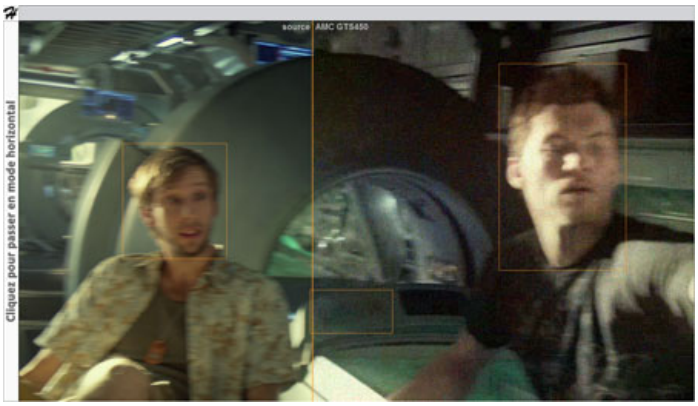
Once again the superiority of the 2p modes is obvious on the graph, but what about the image?

Click here to display the frame comparison in a new tab.

You have to look closely to see the amount of detail and the grain in the skin. The superiority of the double pass modes, including 'veryfast' 2p is there for all to see.

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

In the motion blur scene the results from one preset to another are very close. And how does it look?



Click here to display the frame comparison in a new tab.

As of 'superfast' it does particularly well with very good conservation of detail.

Page 23

StaxRip/x264, Inception/K-On!! 1080p(4/4)

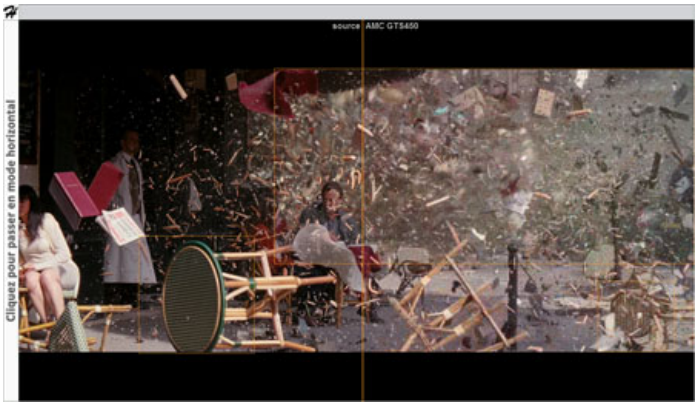
StaxRip/x264, Inception 1080p

StaxRip/x264	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
ultrafast (1 pass)	Baseline	No	1	9178	No	No (I250)	0,952509
superfast (1 pass)	High	Yes	4	9113	Yes	Yes	0,974773
veryfast (1 pass)	High	Yes	4	8794	Yes	Yes	0,976624
faster (1 pass)	High	Yes	4	8872	Yes	Yes	0,9788
fast (1 pass)	High	Yes	4	8890	Yes	Yes	0,978633
medium (1 pass)	High	Yes	4	8876	Yes	Yes	0,978439
slow (1 pass)	High	Yes	5	8923	Yes	Yes	0,978837
slower (1 pass)	High	Yes	8	8912	Yes	Yes	0,979293
veryslow (1 pass)	High	Yes	16	8857	Yes	Yes	0,979285
ultrafast (2 pass)	Baseline	No	1	9934	No	No (I250)	0,959644
superfast (2 pass)	High	Yes	4	9934	Yes	Yes	0,976549
veryfast (2 pass)	High	Yes	4	9934	Yes	Yes	0,979321
faster (2 pass)	High	Yes	4	9934	Yes	Yes	0,981205
fast (2 pass)	High	Yes	4	9934	Yes	Yes	-
medium (2 pass)	High	Yes	4	9934	Yes	Yes	0,980689
slow (2 pass)	High	Yes	5	9934	Yes	Yes	0,981053
slower (2 pass)	High	Yes	8	9934	Yes	Yes	0,981401
veryslow (2 pass)	High	Yes	16	9934	Yes	Yes	0,981476

Now we move on to our most complex scene.

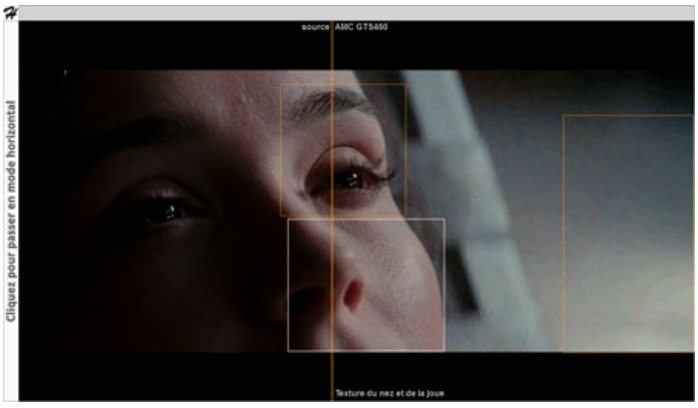
Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

Once again the difference in uniformity in the quality of a single pass scene to a double pass is very clear. Is this as obvious visually?



Click here to display the frame comparison in a new tab.

The 'veryfast' 2p mode already offers excellent quality, a good deal above any of the modes with one pass.



Click here to display the frame comparison in a new tab.

Once again the addition of the second pass is the most important element in maximising the quality of this scene.

K-On !! 1080p

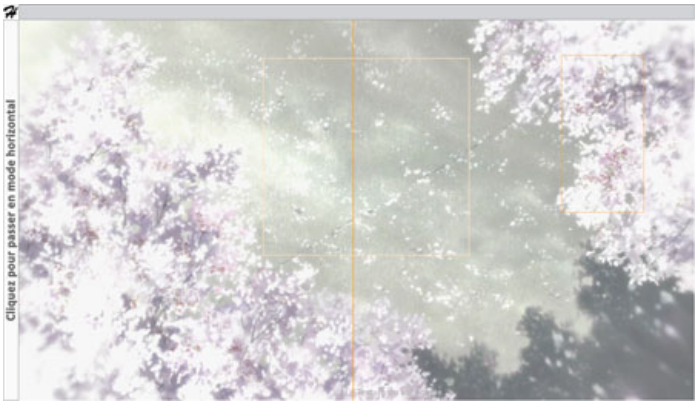
StaxRip/x264	Profile	CABAC	RF	Bitrate	B Frame	Dynamic GOP	SSIM
ultrafast (1 pass)	Main	No	4	10000	No	No (I250)	0,992098
superfast (1 pass)	High	Yes	4	10000	Yes	Yes	0,992242
veryfast (1 pass)	High	Yes	4	10000	Yes	Yes	0,993078
faster (1 pass)	High	Yes	4	10000	Yes	Yes	0,993741
fast (1 pass)	High	Yes	4	10000	Yes	Yes	0,993696
medium (1 pass)	High	Yes	6	10000	Yes	Yes	0,993776
slow (1 pass)	High	Yes	10	10000	Yes	Yes	0,993953
slower (1 pass)	High	Yes	16	10000	Yes	Yes	0,994226
veryslow (1 pass)	High	Yes	16	10000	Yes	Yes	0,994352
ultrafast (2 pass)	Main	No	4	10000	No	No (I250)	0,992488
superfast (2 pass)	High	Yes	4	10000	Yes	Yes	0,992337
veryfast (2 pass)	High	Yes	4	10000	Yes	Yes	0,99309
faster (2 pass)	High	Yes	4	10000	Yes	Yes	0,993763
fast (2 pass)	High	Yes	4	10000	Yes	Yes	0,993717
medium (2 pass)	High	Yes	6	10000	Yes	Yes	0,993795
slow (2 pass)	High	Yes	10	10000	Yes	Yes	0,99397
slower (2 pass)	High	Yes	16	10000	Yes	Yes	0,994242
veryslow (2 pass)	High	Yes	16	10000	Yes	Yes	0,994365

StaxRip/x264	Time	StaxRip/x264	Time
ultrafast (1 pass)	1021	ultrafast (2 pass)	2026
superfast (1 pass)	1045	superfast (2 pass)	2007
veryfast (1 pass)	1122	veryfast (2 pass)	2052
faster (1 pass)	1399	faster (2 pass)	2336
fast (1 pass)	1790	fast (2 pass)	2711
medium (1 pass)	2268	medium (2 pass)	3187
slow (1 pass)	3635	slow (2 pass)	4688
slower (1 pass)	7962	slower (2 pass)	9075
veryslow (1 pass)	11404	veryslow (2 pass)	13586

Encoding times are particularly long here, 'faster' 1p was executed in real time.

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

The scores are once again very high across all modes. Is 'superfast' 1p enough here?



Click here to display the frame comparison in a new tab.

As of 'superfast' 1p the quality is excellent

Click here to display the frame comparison in a new tab.

All modes do a good job here.

Summary

The StaxRip/x264 combination isn't perfect. The addition of a multithreaded decoder would probably improve the speeds of the slowest encode modes. Apart from the competition element however, this probably wouldn't make all that much difference in the end. If you're looking for quality, x264 does the job and if you're willing to add the time required for a second pass, it literally outdoes all the competition, including when you compare a fast double pass mode such as 'veryfast' with one of the slower single pass modes.

Page 24

720p Performance Recap

720p Performance Recap

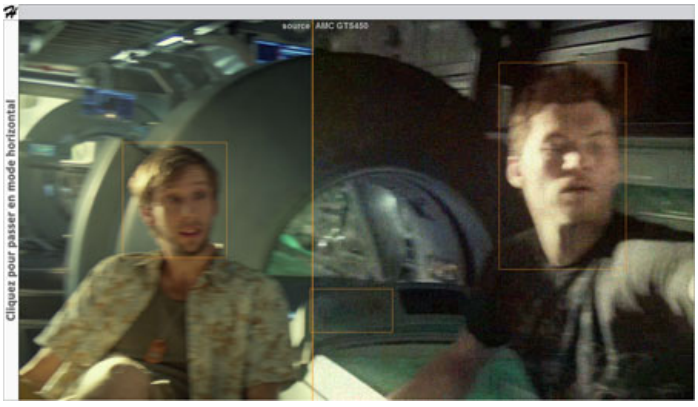
If you want to compare our encoders one against another, we have grouped their results in the following graphs.

Avatar 720p

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

Click here to display the frame comparison in a new tab.

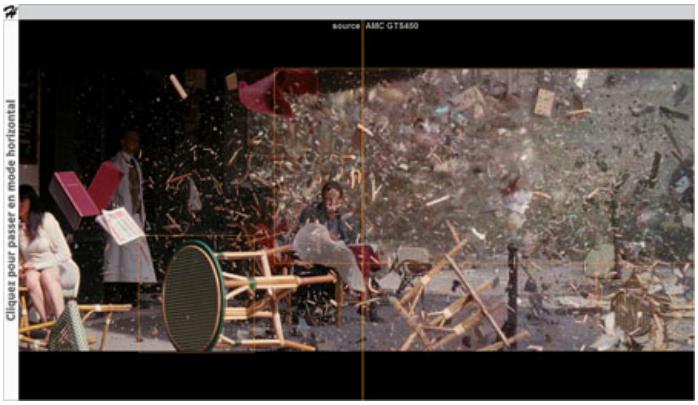
Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.



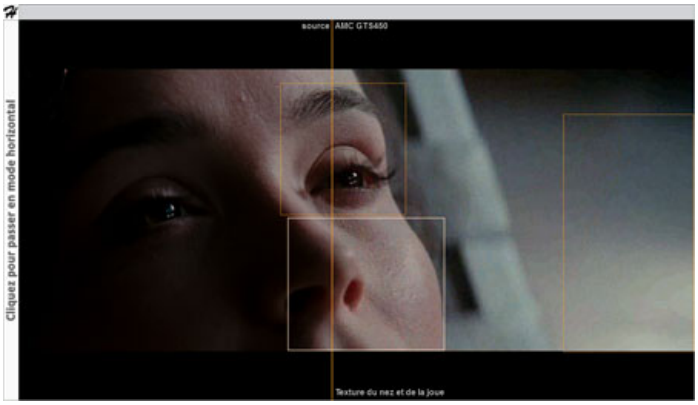
Click here to display the frame comparison in a new tab.

Inception 720p

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.



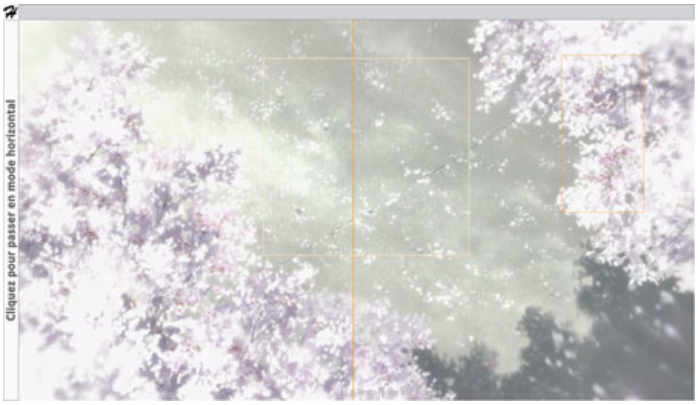
Click here to display the frame comparison in a new tab.



Click here to display the frame comparison in a new tab.

K-On !! 720

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene..



Click here to display the frame comparison in a new tab.

Click here to display the frame comparison in a new tab.

Page 25

1080p Performance Recap

br>

1080p Performance Recap

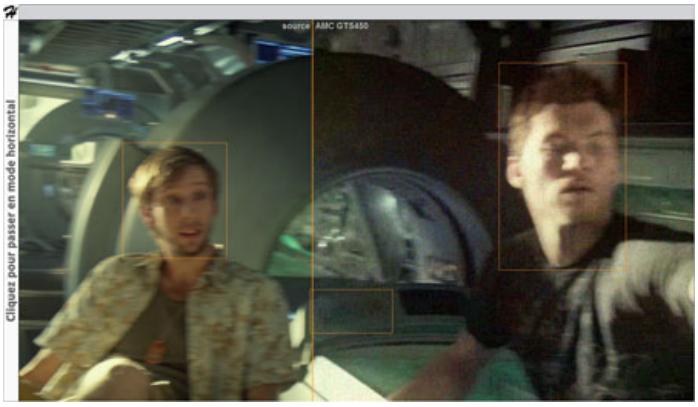
If you want to compare our encoders one against another, we have grouped their results in the following graphs.

Avatar 1080p

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.

Click here to display the frame comparison in a new tab.

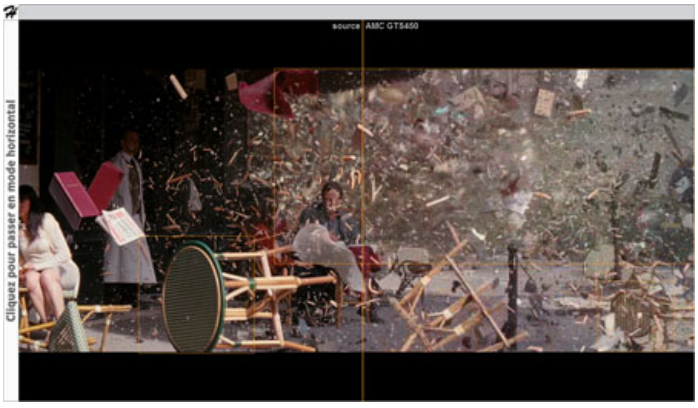
Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.



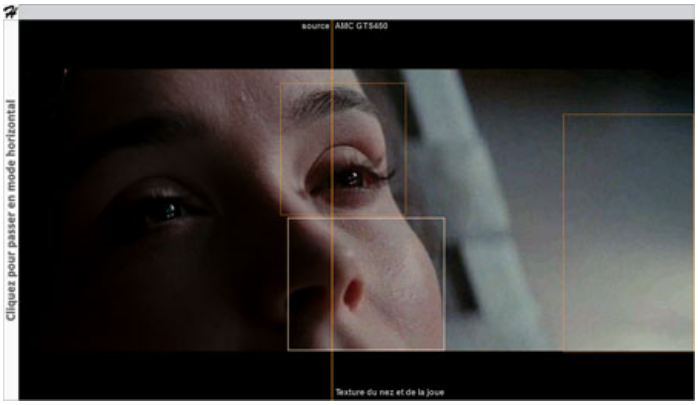
Click here to display the frame comparison in a new tab.

Inception 1080p

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.



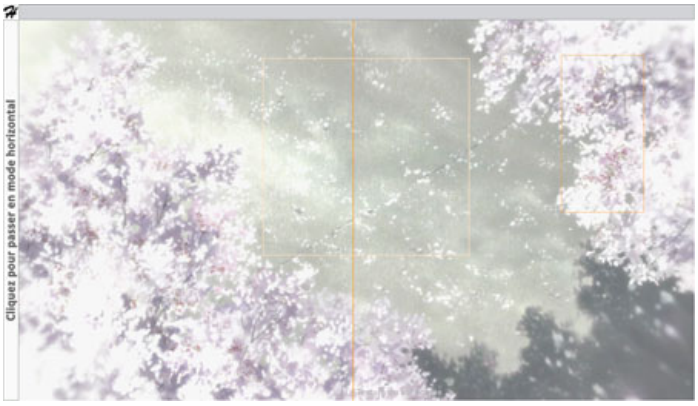
Click here to display the frame comparison in a new tab.



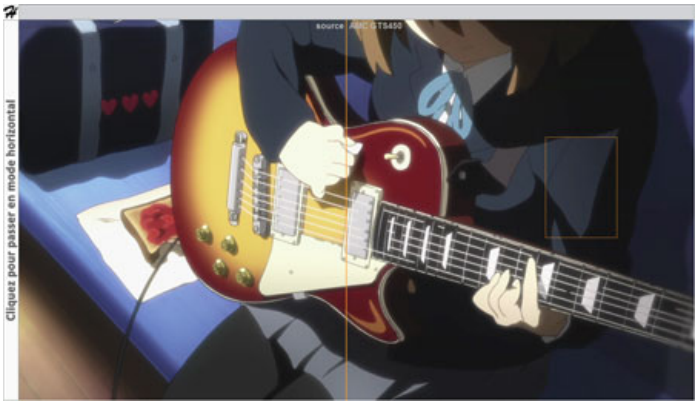
Click here to display the frame comparison in a new tab.

K-On !! 1080

Use an HTML5 compatible browser to view the graph!
Click here to view the PSNR graph of this scene.



Click here to display the frame comparison in a new tab.



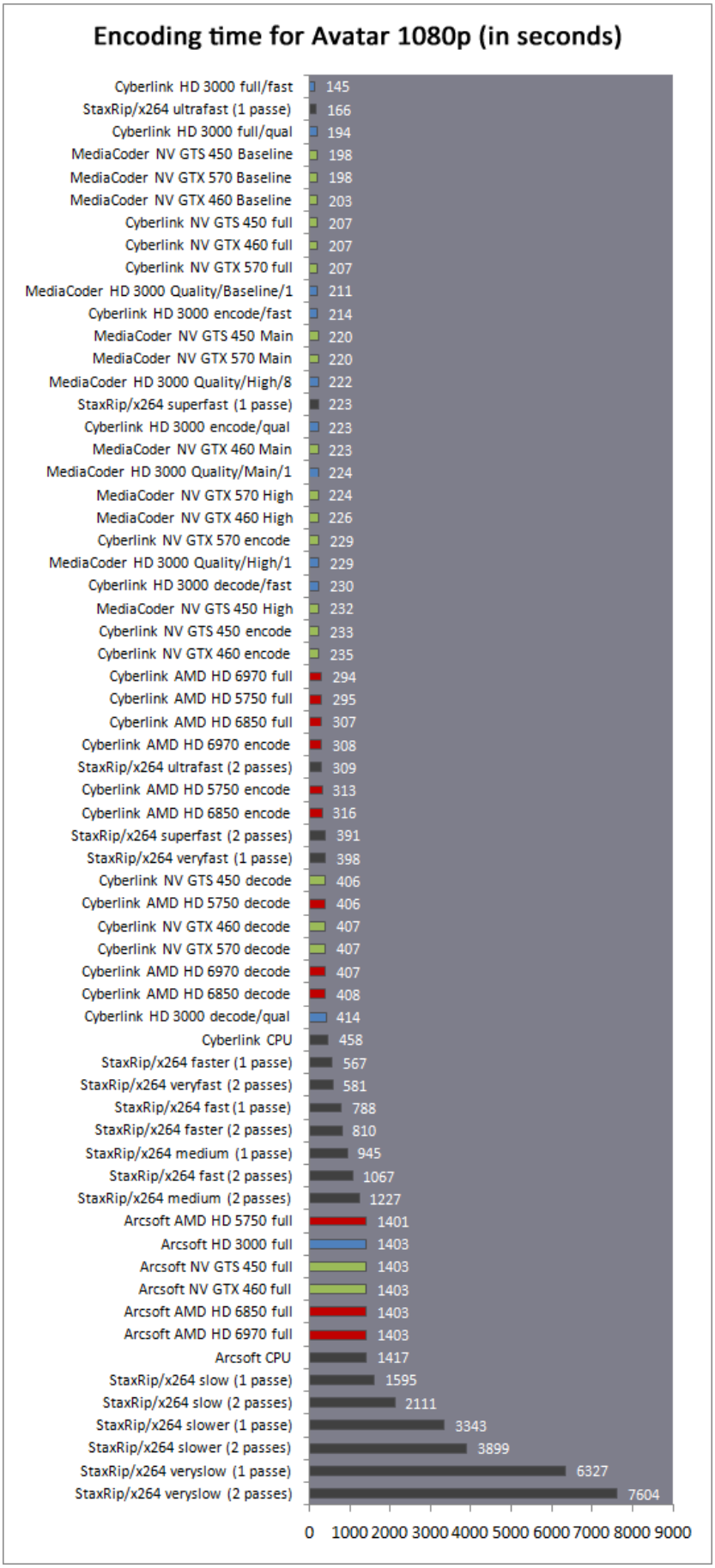
Click here to display the frame comparison in a new tab.

Page 26

Energy consumption/Time Recap

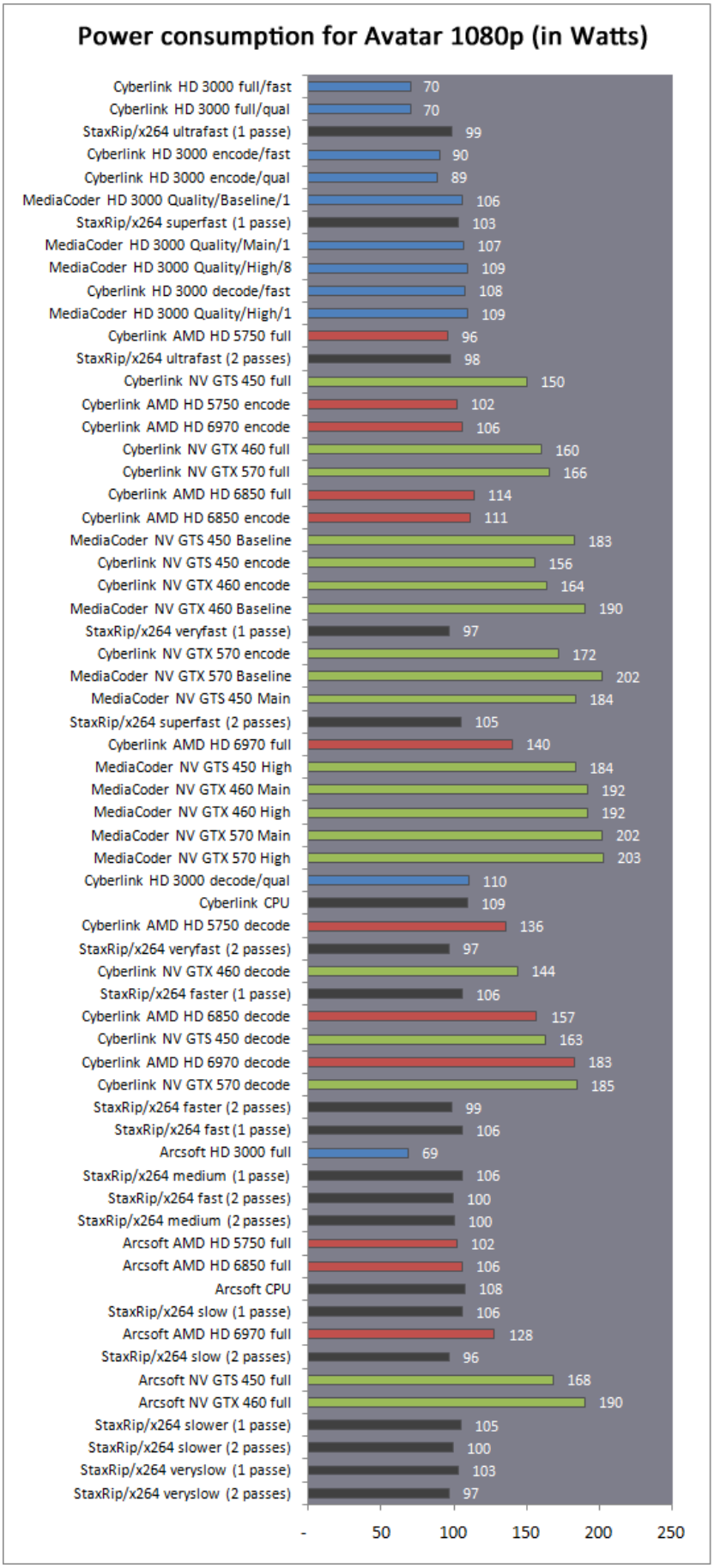
Energy consumption/Time Recap

Here are the transcoding times for each of the encoders tested in this report:



To recap, the encoding time for the Arcsoft encoders is particularly slow in this scene, probably because of the transcoding of the audio track, which seems to be a problem in this application. In terms of pure rapidity, the HD 3000 encoder is well placed, as are the NVIDIA via MediaCoder encoders. x264 can be fast, but its 'ultrafast' mode quality is unusable to our taste.

Hold the mouse over the graph to see energy consumption in Watts.



We have calculated the real consumption of each encoding by multiplying the encode time by the energy consumption. The results are quite interesting and bring a few things to mind:
-If you look at the energy consumption alone, you'll see that the NVIDIA graphics cards consume most energy when running. This is no surprise as these cards go into 3D mode during encoding (to what end?).
-The HD 3000 solutions are by far the most efficient and take most of the top spots. x264 'superfast' is also pretty well placed. To our mind, 'veryfast' 2p is the best compromise between quality, encoding time and energy consumption.

Page 27

Conclusion

Conclusion

It's difficult to provide a succinct conclusion to so many tests, but one point does stand out: GPU acceleration of H.264 transcoding isn't on a par with encoding carried out by CPUs.

What these solutions bring most of all is frustration. Whether NVIDIA, AMD or Intel solutions, rapidity has been accentuated to the detriment of quality. It's rather surprising to see that with software such as MediaConverter from Arcsoft or MediaEspresso from Cyberlink, CPU encoders systematically give a better result in terms of quality than the integrated GPU encoders!

It's also extremely annoying to note that in the case of GeForce and Radeon encoding, there's no difference in speed between graphics cards costing 100, 170 or 330 euros. Quality is strictly identical from one card to another – except in the case of bugs – and encoding times are no different from one card to another either. In no case was the GPGPU power of our graphics cards fully used.

The use of graphics cards in such tasks still requires the help of a CPU. Even when GPU decoding and encoding are both used, with Cyberlink for example, CPU core occupation is 100% for one core with the Radeons or the Intel HD 3000, 100% for two cores with the GeForces and as many as four cores at 100% occupation with Arcsoft.

Another important particularity is that the decoding carried out by a GPU often puts a brake on the performance of GPU encoders. Once again, this is logical: H.264 decoding isn't carried out by the GPU's processing units but by a dedicated ASIC which serves to decode videos such as Blu-rays. This decoding doesn't need to be done extremely fast as the playback of such media is in real time. In spite of its faults, MediaCoder proves that to get the most out of GPU encoders in terms of speed (without, for all that, creating any variation between our differently priced cards…), you have to use a multithreaded CPU decoder.

It's difficult to recommend any one of the three solutions using GPUs for transcoding over the others. The Arcsoft application offers the encoder which, apart from x264, gets the best scores and is also extremely fast. Visually, results with the Arcsoft encoders are blurred but they may be sufficient for mobile peripherals if you're not too fussy. These results were however obtained solely with the Media Converter CPU encoder, its CUDA version literally being a nightmare and its Radeon version, even though producing higher quality, being limited (like the rest) to a baseline profile which can't compete with the highest H.264 profiles. Moving onto the Intel/MediaSDK version, although it obtained high SSIM and PSRN scores, it doesn't measure up to the naked eye and important parts of the image (faces and so on) are very blurred.

The Cyberlink application rules itself out in only offering baseline type H.264 encoding, which isn't up to handling action scenes. The numerous implementation bugs in the Cyberlink and MediaSDK software doesn't help either (white and black squares make the encoded files unusable) and the fact that the encoders are unable to insert an I-frame when a new scene starts creates disagreeable flickering. The quality of the NVIDIA and AMD renderings is okay, which does at least represent some progress on the Arcsoft application for the GeForces. In practice however, the use of baseline H.264 profiles remains a handicap that it is impossible to compensate for.

MediaCoder is the fastest, the most configurable and the most efficient of the GPU encoding applications. If you can stand the advertising however, the fact that the files it produces lack frames is nonetheless a bit of an issue. At least it's free. From a qualitative point of view, the NVIDIA encoder has the advantage here over the Intel, which tends to blur frames a little too much.

The StaxRip/x264 combination wins hands down for quality. With an equal number of passes, the 'faster'/'fast' modes generally do as well, if not better, than the rest of the encoders tested here. If you only retain one thing from this article however, make sure you remember that the simplest way of increasing quality is simply to add a second pass. It is all the more annoying that NVIDIA, AMD and Intel could easily offer this second pass in their development kits and thus miraculously homogenise quality throughout their encoded videos.

Of course, you pay for this superior quality with higher encoding times, equivalent to real time at 1080p (one hour of encoding for around one hour of film in 'veryfast' 2p mode with Avatar for example). You can reduce quality too. Some users will find the 'faster' 1p modes give a more attractive compromise by more or less halving encoding time, but below this quality really does start to suffer.

In trying to sum up what AMD, NVIDIA and Intel are offering via these third party applications, we do owe it to ourselves to make a few remarks. Firstly, the AMD encoder is anything but stable. The applications that it was running in crashed on numerous occasions, something that we were able to reproduce when launching the AMD transcoding interface manually (you can access it via the CCC control panel). In limiting the encoder to the baseline profile here again, AMD isn't giving itself any real opportunity in terms of decent quality overall. This is particularly regrettable as in static scenes the quality is often pretty good. Implementing a high H.264 profile would be a good idea.

NVIDIA possibly has the most advanced SDK and its results are often the best when it comes to GPU acceleration. Nevertheless, the visual quality remains poor, to the point where the pure CPU solutions are often able to give an equivalent quality/encoding time ratio. What's more, power consumption of these graphics cards is very high.

The Intel offer is the most surprising of the lot. The video encoding is managed by units added to the GPU which seem to accelerate a large part of the decoding. With very low CPU occupation, energy consumption is by far the lowest. In terms of issues with the Intel solution, you need to have an H67 or Z68 motherboard to run it, which greatly reduces the potential user base. Even if you do have one of these however, the visual quality of the encoded files frankly leaves too much to be desired for it to be really usable.

At the end of the day, the marketing promises in terms of GPGPU transcoding haven't been kept. The manufacturers highlight the rapidity of their solutions as a solution to the very real problem of the excessive amount of time required for CPUs to encode video alone. By offering rapid encoding solutions, but with quality that leaves too much to be desired, H.264 encoding via GPGPU solutions remains, as yet, a poor solution to what is a real problem.

FFMPEG with CUDA acceleration

Bids	Avg Bid (USD)	Project Budget (USD)
0	N/A	$30 - $5000

CANCELLED

Project Description:
Hello, I need help to use my FFMPEG schemes to be accelerated and speed up the processing time.

Deliverables

What process to speed up and what schemes, to know about them, please mention it your message to me. This job is only for the coders who have done this before, exactly.

Skills required:
C Programming

See more: ffmpeg cuda, speed ffmpeg cuda, ffmpeg cuda acceleration, acceleration, ffmpeg

Post a Project like this

Project posted by:
vagabonds
5.0 (11 Reviews)
VERIFIED

Follow

Project ID: 2782869
Report Project

Public Clarification Board

0 messages

Freelancers Bidding	Reputation	Bid (USD)

Video decoder on Cuda ffmpeg

1

I starting to implement custum video decoder that utilize cuda HW decoder to generate YUV frame for next to encode it.

How can I fill "CUVIDPICPARAMS" struc ??? Is it possible?

My algorithm are:

For get video stream packet I'm use ffmpeg-dev libs avcodec, avformat...

My steps:

1) Open input file:

```
avformat_open_input(&ff_formatContext,in_filename,nullptr,nullptr);
```

2) Get video stream property's:

```
avformat_find_stream_info(ff_formatContext,nullptr);
```

3) Get video stream:

```
ff_video_stream=ff_formatContext->streams[i];
```

4) Get CUDA device and init it:

```
cuDeviceGet(&cu_device,0);
CUcontext cu_vid_ctx;
```

5) Init video CUDA decoder and set create params:

```
CUVIDDECODECREATEINFO *cu_decoder_info=new CUVIDDECODECREATEINFO;
memset(cu_decoder_info,0,sizeof(CUVIDDECODECREATEINFO));
...
cuvidCreateDecoder(cu_video_decoder,cu_decoder_info);
```

6)Read frame data to AVpacket

```
av_read_frame(ff_formatContext,ff_packet);
```

AND NOW I NEED decode frame packet on CUDA video decoder, in theoretical are:

```
cuvidDecodePicture(pDecoder,&picParams);
```

BUT before I need fill CUVIDPICPARAMS

CUVIDPICPARAMS picParams;//=new CUVIDPICPARAMS; memset(&picParams, 0, sizeof(CUVIDPICPARAMS));

```
HOW CAN I FILL "CUVIDPICPARAMS" struc ???

typedef struct _CUVIDPICPARAMS
{
    int PicWidthInMbs;      // Coded Frame Size
    int FrameHeightInMbs;   // Coded Frame Height
    int CurrPicIdx;         // Output index of the current picture
    int field_pic_flag;     // 0=frame picture, 1=field picture
    int bottom_field_flag;  // 0=top field, 1=bottom field (ignored if field_pic
    int second_field;       // Second field of a complementary field pair
    // Bitstream data
    unsigned int nBitstreamDataLen;        // Number of bytes in bitstream data
    const unsigned char *pBitstreamData;   // Ptr to bitstream data for this pic
    unsigned int nNumSlices;               // Number of slices in this picture
    const unsigned int *pSliceDataOffsets; // nNumSlices entries, contains offse
    int ref_pic_flag;       // This picture is a reference picture
    int intra_pic_flag;     // This picture is entirely intra coded
    unsigned int Reserved[30];             // Reserved for future use
    // Codec-specific data
    union {
        CUVIDMPEG2PICPARAMS mpeg2;          // Also used for MPEG-1
        CUVIDH264PICPARAMS h264;
        CUVIDVC1PICPARAMS vc1;
```

tagged

cuda × 6185
ffmpeg × 4989
nvidia × 942
libav × 161
decoder × 146

asked **9 months ago**
viewed **1324 times**
active **9 months ago**

Related

2 Modifying motion vectors in ffmpeg H.264 decoder

0 how to decode wmp3 video using libavcodec? (ffmpeg)

15 Decoding Video using FFMpeg for android

1 CUDA H.264 decoder initialization problems

3 Decode a video file from memory using libav

0 How get sps struc from h264 video with libav ffmpeg

0 Video decoding using ffms2 (ffmpegsource)

0 ffmpeg error on decode

1 FFMPEG: Working of parser of a video decoder

Hot Network Questions

Why there is such a big difference between "Size" and "Size on disk"?

How do I explain to a 6 year old why people on the other side of the earth don't fall off?

The Spiral of Roots in TikZ

What is the difference between

```
        CUVIDMPEG4PICPARAMS mpeg4;
        CUVIDJPEGPICPARAMS jpeg;
        unsigned int CodecReserved[1024];
    } CodecSpecific;
} CUVIDPICPARAMS;

typedef struct _CUVIDH264PICPARAMS
{
    // SPS
    int log2_max_frame_num_minus4;
```

cuda ffmpeg nvidia decoder libav

share | edit | flag

asked Apr 25 '13 at 19:37
Oleksandr Kyrpa
24 ●4

possible duplicate of Decode video with CUDA nccuvid and ffmpeg – talonmies Apr 25 '13 at 20:25

You already asked this question. Please don't ask it again. – talonmies Apr 25 '13 at 20:51

Have you looked at the NVIDIA video decode samples? – Robert Crovella Apr 25 '13 at 20:55

Yes I use example from CUDA SDK. But it uses proprietary video file parser. – Oleksandr Kyrpa Apr 25 '13 at 23:51

add comment

1 Answer

active | oldest | votes

2

This is the purpose of the `CUvideoparser` object. You feed it the data stream frame by frame through `cuvidParseVideoData`, and it calls you back with `CUVIDPICPARAMS` ready to pass to the decoder when it detects it has a complete frame ready.

All this and more is very well illustrated in the D3D9 decode sample, available here. I suggest studying it in detail because there's not much documentation for this API outside of it.

share | edit | flag

answered Apr 25 '13 at 21:13
Asik
5,793 ●1 ●11 ●44

Yes I know and use D3D9 decode sample, but I dont use CUvideoparser! I use ffmpeg -> AVpacket to get stream packet data from video source. So I need correct fill CUVIDPICPARAMS. How I did know codec spec from ffmpeg to parse it in CUVIDPICPARAMS struc? – Oleksandr Kyrpa Apr 25 '13 at 23:49

I'm not too familiar with FFmpeg, but I'd assume you can simply pass the data of an AVpacket to CUvideoparser and it'll fill CUVIDPICPARAMS for you. Is there any reason why you'd not use it? – Asik Apr 26 '13 at 0:05

CUvideoparser not read mkv video files for me, different files -different containers – Oleksandr Kyrpa Apr 26 '13 at 0:24

But in case if I use CUvideoparser app will crash on cuvidParseVideoData(pParser,&cu_packet) step. – Oleksandr Kyrpa Apr 26 '13 at 0:27

MKV has nothing to do with CUvideoparser, it's a container. What you give to CUvideoparser are video frames of a given encoding (i.e. H264, MPEG2, etc), it doesn't matter where they come from. – Asik Apr 26 '13 at 0:35

add / show **4** more comments

Your Answer

Links Images Styling/Headers Lists Blockquotes Code HTML advanced help »

community wiki

a point and a vector

Is it ethical to write answers to work-relevant Stack Exchange questions on the clock?

Raising the flaps right after touchdown. Good or bad?

Should "Yes, delete it" be red or green?

Hello world! with limited character repetition

Cheating in a phone interview

Isolated space colony restricted to a very small mountainous area of an otherwise uninhabitable planet

Meaning of "Christians should think about the humility, poverty and simplicity of the birth of Jesus Christ"

Carrying medicines internationally for a friend

Temperature conversion in C

Environment that doesn't allow blank lines

Does the C preprocessor remove instances of "&*"?

How to calculate with absolute value.

Postdoc salary. Fair offer?

Translation of "I can't eat spicy food"

Can a program tell it is being run under sudo?

How to voice an NPC?

is there any significance to using tee

What is a 'friendly' way to let managers know that having good developers is a privilege?

Is it safe to write the salt AND/OR the IV at the beggining of an encrypted file?

Automorphisms of Surfaces, Open Books and Contact Structures

Post Your Answer

Not the answer you're looking for? Browse other questions tagged cuda ffmpeg nvidia decoder libav or ask your own question.

question feed

The wretched state of GPU transcoding

By Joel Hruska on May 8, 2012 at 1:19 pm | Comment

1 of 9 | Next »

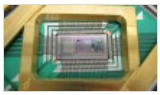
This story began as an investigation into why Cyberlink's Media Espresso software produced video files of wildly varying quality and size depending on which GPU was used for the task. It then expanded into a comparison of several alternate solutions. Our goal was to find a program that would encode at a reasonably high quality level (~1GB per hour was the target) and require a minimal level of expertise from the user.

It's been several years since Nvidia began pushing Badaboom and the idea of GPU-assisted transcoding, and given Intel's major entry into the market last year, we expected to find a crop of mature, effective solutions. Cyberlink's MediaEspresso and Arcsoft's Media Converter aren't new products; the latter is often recommended by Intel and Nvidia as a way to see the potential of Quick Sync/GPU transcoding. Users are posting more video of themselves on social networks and the camera quality of cell phones is a major buying point for a lot of people — so what software solutions work best?

The primary comparison here is between Xilisoft's Ultimate Video Converter, Arcsoft Media Converter, and Cyberlink's MediaEspresso. Other potential options, including Avivo, Badaboom, and MediaCoder are discussed at the end of the article. Initially, the goal was to include quality assessments on three separate sources, but none of these three solutions were able to properly encode our 30GB Blu-ray rip of *Lord of the Rings: The Return of the King*. MediaEspresso was able to render the video but not the audio, Arcsoft simply crashed, and Xilisoft's audio was distorted. Data on the encoded file (when one was created) is still included here as it sheds light on how these programs handle different workloads.

MediaCoder was the one application capable of properly encoding *Return of the King* properly, but it's not on our list of direct comparisons. Our goal was to review consumer-oriented packages that offered simple presets; MediaCoder is immensely capable, but thoroughly intimidating for anyone who doesn't have an intimate understanding of the encoding process. We have included screenshots from both the source material and Handbrake as a default, CPU-driven video encoder.

We'll be comparing the encode quality of two test files. The first is a standard DVD rip of the movie *Stargate: Ark of Truth.* It's a bog standard 720×480 DVD laid down using MPEG-2; we encoded the first 20 minutes of the movie (one VOB file). The second is an already-encoded version of the *Star Trek: The Next Generation* episode "The Inner Light."

More Articles

New benchmarks raise doubt over D-Wave's 'quantum computer,' but Google is optimistic long-term Jan 20

Please, Microsoft, don't put Windows XP to sleep on April 8 – the world isn't ready yet! Jan 20

It's time for AMD to take a page from Intel and dump Steamroller Jan 20


The Wii U is dead in its current form, admits Nintendo. But what now? Jan 20

Restoring the function of arms that have been disconnected from the brain Jan 17

The TL;DR version of this article is as follows: If you want a video encoder that'll run on virtually any system, has well-thought, easy-to-use presets, and neatly balances quality and file size, go download Handbrake. It's fast, free, and efficient — it just doesn't use the GPU.

The best we can say of these three is that some of them do a decent job, on some hardware, some of the time.

1 of 9 | Next »

Photos: 1 2 3 4 5 6 7 8 9

Tagged In

Software Slideshow Intel Amd Featured Gpus Graphics Cards Ivy Bridge

Share This Article

Post a Comment

Comment

- *http://geek.com* *sal cangeloso*

  Incredible work Joel!

  I've always been a Handbrake fan, so it's great to see that I (probably mostly by accident) have been using the right tool for the job. Sometimes I do us Adobe Media Encoder, but only for specific jobs.

  That said, I avoid transcoding at almost all cost, even if that means going without video to watch.

- *http://www.mrseb.co.uk/* *Sebastian Anthony*

  Yeah, I use Adobe stuff — Premiere, and then Media Encoder. Fairly easy to use — some nice presets that are easy to tweak.

- *http://www.facebook.com/profile.php?id=1223563048* *Angel Ham*

  GPU and CPU transcoding feels like sex:

  One delivers quicker, albeit unsatisfying results whereas the other one requires patience but the outcome (I see what I did there) is much more satisfying.

- *Joel Hruska*

  snort

  There's nothing inherently wrong with GPU encoding. Arcsoft's Quick Sync encodes look great. So do Xilisoft's CUDA encodes. The problem is with the software, not the hardware.

- *Ron Johnson*

  But since you need software to actually use the hardware, then there actually is something wrong with GPU encoding.

- *http://twitter.com/zervin* *Zack Ervin*

  What about DVDFAB.., I've been using it for awhile and can regularly pull several hundred frames per second. Couple this with something like a more advanced CoreAVC codec and the output is pretty good and pretty stable.

- *Joel Hruska*

  Zack,

  I have no doubt that there are other good programs out there. The complexity of testing, however, prevented me from including more than a few select options.

  If you're comfortable working without presets, MediaCoder, for example, can give excellent GPU results. The problem (or at least, the difference) is that MC has no learning curve. There's an iPhone edition, but it forces a 960×640 resolution by default and produces garbage if you attempt to force a custom resolution that takes the source's aspect ratio into account.

  I chose the programs that a user was most likely to encounter after doing quite a bit of Googling and forum searching for what users recommended.

- *William Robbins*

  The graph on page two is incorrect, you duplicate the same graph twice. Editing people.. attention to detail. :/ I would really like to see the comparison there for myself. Unfortunately it detracts from my confidence in the rest of the article, even though I know it was a mountain of work to create.

- *http://www.mrseb.co.uk/* *Sebastian Anthony*

Sorry, that was my fault — will fix; thanks!

- *Joel Hruska*

William,

In this case, during the editing, one graph's data got duplicated over the other. Thank you for bringing this to our attention; I corrected it several hours ago. The proper result should now be synchronized with the database.

- *coburndomain*

Have you guys considered Any Video Converter? It apparently can do CUDA-accelerated video encoding. Haven't tried it, though.

- *Joel Hruska*

Coburn,

The programs listed in the article are the ones I tried. I never came across any references to Any Video Converter.

- *http://pulse.yahoo.com/_B3KQHHSRLCQFZE3UNIGQBS3QIE Cool Mike*

I think this is a software issue. I use a MAC so I think Open CL is the way to go (Apple GL)

- *http://profile.yahoo.com/5ICDDOUSQUVQ5F3YZAFG6DZKEQ Virgilio C*

Still Avisynth with MeGUI or X264 are the best

- *drich44*

Freemake uses Cuda, does a very good job of reducing files, and has super easy presets for almost any device. You can also customize output if you wish.

- *jason leblanc*

A friend of mine got me to try Handbrake a few months back but I wasn't really impressed (at least not by its encoding speed). Most transcoding I do is to put video files into formats more suitable for my iPod Touch, and for the past few years I've been using iPodME. It's a fairly simple transcoder program that I think uses FFDshow or something.

A few years later, I've still not found a program that transcodes faster then iPodME. It even scales nicely when you oc your cpu. OC by 25% and you'll basically get a 25% increase in transcoding fps.

GPU transcoding has always been abysmal, at least for me and the ATI cards I've tried it with.

- *http://profiles.google.com/jhancockdarwin James Hancock*

Now try something hard like recoding a 1080p movie and retaining/converting the DTS audio tracks so that you end up with H264 + DTS + AC3 for fall back…

And let's make it a little harder and require it to play back on an Xbox 360 with full 1080p AND 5.1 sound.

Only one that comes close is DVDFab and even that crashes on a regular basis on wild MKVs.

And it's Intel Quicksync with z68 etc. with Virtu crashes the application and it doesn't support AMD other than through DVXA which only decodes.

When are we going to see one app that:

Can convert to WMV VC1 1080p with 5.1 Pro, M2TS 1080p with DTS and AC3 (both copy DTS and transcode it down to AC3), encode MKV with DTS and AC3 that creates video streams that are not broken, and can encode and .mp4 file with 5.1 aac or AC3 audio that actually works?

And can actually accept essentially all formats?

Even MediaCoder doesn't work with AAC 5.1 (M4A) audio sources properly, and is virtually impossible to transcode to M2TS which other than WMV with VC1 and WMA Pro is the only container/format that Xbox and others like.

Sorry but Handbreak is NOT FAST. We're talking several hours to encode a movie at 1080p and it's defaults are not compatible with Xbox 360 and it routinely eats 5.1 audio into Stereo even when you have it set to encode to AC3 or even pass through DTS.

It would be great if the author would do a more extensive test and take a BluRay and rip it for storage for your media center (plex, wmc, myth, whatever) in the same resolution with audio formats that can be decoded in high res with extenders working too.

Then test. These companies need to be named and shamed for their crappy performance. Only exposees like this will embarass them into fixing their software so that it works and makes logical sense!

- *tpjo*

error reply, sorry.

- *Joel Hruska*

James,

After the substantial difficulty of testing *simple* software doing basic tasks, I'm a bit burned out on the idea of actually doing anything complicated.

I had a bit of trouble with Handbrake inserting skips into one file I tested it with in an attempt to synchronize audio and

video. I was never able to successfully resolve it, though it was a corner case.

To be perfectly frank, these are the sorts of issues that are extremely difficult for anyone to tackle in an organized way. Proper evaluation requires an enormous amount of preparation and care to ensure that results are accurate.

- *konran*

Interesting, but I'm wondering why you haven't included TMPGEnc Video Mastering Works 5 in your tests?! It is capable to transcode with CUDA with good results – but unfortunately I don't have comparison results to other tools.

Limitations: There is no AMD support and High Profile cannot be used with CUDA. Therefore High Profile is limited to CPU transcoding usage.

- *Joel Hruska*

Konran,

You answered your own question. :)

- *http://twitter.com/timverry Tim Verry*

Props on using a Stargate movie, the article is already off to a good start and I'm only on page 1 ;)

- *kukreknecmi*

All theese programs rely on developer lib. The transcoder dll do the most of the job. They all sacrifice great amount of quality. All of them do the compression unefficient 1pass scheme. Even the 4gb file feels like acceptable cuz all of our perception changed with years and we dont care about how inefficient the compression is. Years back, there was great concern about trying to keep the most of the quality. All theese type of transcoders mean smthg if you need a quick conversion. You need to sacrifice either quality or storage space.

The only real thing at hand is 2p hi profile encoding, if you really want to rip from 30gb to 2-4gb. The best tool for this is x264 with some frontend (like MeGui).

What is lacked and has been requiring for years is GPU accelerated 2p hi profile encoding. Smthg like x264 + Open CL with 2p slowest preset and hi-profile encoding. After then we can start talkin about a real re-encoded hi-quality rip/video.

Check this great article:
http://www.behardware.com/articles/828-1/h-264-encoding-cpu-vs-gpu-nvidia-cuda-amd-stream-intel-mediasdk-and-x264.html

- *http://geek.com sal cangeloso*

Thanks for the link. Cool post

- *http://profile.yahoo.com/VBE6A6QIJDVFZOMPVGXJCY2AHU Max*

Curious, how come Freemake was not included in the review?
http://www.freemake.com/
I've been using it for the last 3 months and it has scaled very well from AthlonXP 3000 no GPU acceleration all the way to Core i7 (Hyper threading enabled) with Nvidia Quatro FX 1800 and has been able to handle just about any type of video that I threw at it.

- *Joel Hruska*

Max,

Here's how the various programs were selected. Note that I spent quite a bit of time with certain packages that ultimately weren't included. Here's the breakdown:

Cyberlink MediaEspresso: Intel, Nvidia, AMD recommended.
Arcsoft Media Converter: Intel, Nvidia recommended.
Avivo: Already aware of, largely incompatible.
Badaboom: Already aware of, discontinued, audio problems
MediaCoder: Found via Google search
Xilisoft: Found via Google search.

I searched for GPU encoders, GPU transcoders, GPU encoding software, Quick Sync encoder, and a number of permutations on those terms. I went with the software that was mentioned most often in forums or that came up in discussions.

Freemake, like DVD Fab, is a program that people apparently use but that didn't make it across my radar. I will say that I was aiming to find programs that supported at least two of the three GPUs. I *thought* Xilisoft would run on the latest Radeon — if I'd realized that it wouldn't before I started work, I would've picked a different program.

- *me me*

Any one try the Corel Video Studio and the nero equilant?

- *John Pombrio*

After converting over 1,100 movies, I am simply amazed on how good Handbrake works. Movie files are around a Gig each, look terrific on a 40 inch 1080P TV, have 5.1 surround sound, and can easily be streamed over a wired network. To do a movie takes 8-10 minutes or so. Why on earth would anyone use anything else?

- *Dromo*

It really is laughable .. much trumpeted but MIA (HW assisted encoding). I believe that an early version of BadaBoom killed my Video card. It was Beta at the time, but kinda puts you off experimenting.
OTOH if you want to spend 4/5 hours encoding to a lower quality, larger size file then thse are the softs for you. Surely anyone would be entitled to their money back on this stuff?

- *Joel Hruska*

Software doesn't kill hardware. Ever. If your video card overheated, it's because of a poor cooler design or possibly dust buildup. Even then, I'm extremely dubious. Modern cards down-throttle to prevent this from occuring, the same way CPUs do. They've also got alarms built in.

- *Terry*

Software could, however, kill firmware.

- *Joel Hruska*

How?

Unless you're referring to a bad BIOS flash or something of that nature?

- *olyteddy*

Another up and comer is Mirillis. I use their Action! screen recording software which records in a mildly compressed format (approx. 15GB per hour for 720p) and transcodes a 2 hour recording to a 4GB per hour MP4 in about 20 minutes using Quick Sync on my Intel i7 2600. I believe their Splash! EX program uses Quick Sync encoding too, so I would expect similar results.

- *Horus Delta*

sprash ex pro and cuda is very cool software

- *http://twitter.com/Macross9321 raymond Causwell*

thank you was searching high and low for a detail case study like this – just saved me upgrading my video card

- *someone_asdf*

I don't think the author of the article realizes that he's specifically picked an encode size of 3.5″ due to his choice in profiles.

The smoke blockiness is irrelevant, because – at that screen size – it won't be very legible.

Sure, on a 25″+ TV/display, it's going to look like garbage. On a 3.5″ display? It's going to look absolutely fantastic. The QuickSync did exactly what it was expected — encoded to the specified profile's capabilities.

- *u-man*

sorry to say that, but you have wasted your time with programs, which i would never speak about or dare to use it.

like some people sayed allready, i would recommend DvDFab for the not so experienced user and if you would like to do some Pro stuff then go with Mainconcept Broadcast Suite and Premiere. if you have both, you can do nearly everything. results are high-quality standard, no blockiness no artifacts no nothing. i can speak only for CPU and CUDA stuff, but there is absolutely no reason to say "GPU transcoding (still) not ready for prime time"… this dude, is total bullshit. i did encodings over 6 months in 2011 with both programs, there was no problems at all, even stereoscopic stuff. our clients (on demand and broadcast clients) never send a file back, or bitching on quality-problems.basically you tested crappy software which is not worth talking about. its still good for other people to know, that spending money on this crap is not a solution.rather spend the 200$ and have a complete software product, with no problems at all.http://www.dvdfab.com/all-in-one.htmmy 50cents

- *androidmagic*

Looking at the Star Trek images, the GPU rendered version by Xilisoft is actually clearly superior to the CPU rendered version. It has much better sharpness and recreation of the detail than the CPU version. Right now I have no problem with the conversion size as you can set it manually if you don't want standard output. CUDA-based assistance is much, much faster than just CPU based and I don't know what that application does but it actually works. It does have some other glitches at times (like hard encoding subtitles that I can't seem to remove without removing the stream entirely) but it does a fine job. DVDFab doesn't work for me with GPU assistance. Sony Vegas makes good use of CUDA too and it usually works well for rendering. But I do agree that with most things it's not as great as I expected. Now nVidia has a new chip out of rendering and supposedly it's clearly worse quality than QuickSync and CUDA.

- *Joel Hruska*

Yes, the GPU version of Xilisoft is better than the CPU version, because the CPU version looks terrible. Also, it injects additional frames. The ability to hand-tune an encoder for good output wasn't the focus here.

FFmpeg and x264 Encoding Guide

x264 is a H.264/MPEG-4 AVC encoder. The goal of this guide is to inform new users how to create a high-quality H.264 video.

There are two rate control modes that are usually suggested for general use: Constant Rate Factor (CRF) or Two-Pass ABR. The rate control is a method that will decide how many bits will be used for each frame. This will determine the file size and also how quality is distributed.

If you need help compiling and installing libx264 see one of our FFmpeg and x264 compiling guides.

Contents

Constant Rate Factor (CRF)

This method allows the encoder to attempt to achieve a certain output quality for the whole file when output file size is of less importance. This provides maximum compression efficiency with a single pass. Each frame gets the bitrate it needs to keep the requested quality level. The downside is that you can't tell it to get a specific filesize or not go over a specific size or bitrate.

1. Choose a CRF value

The range of the quantizer scale is 0-51: where 0 is lossless, 23 is default, and 51 is worst possible. A lower value is a higher quality and a subjectively sane range is 18-28. Consider 18 to be visually lossless or nearly so: it should look the same or nearly the same as the input but it isn't technically lossless.

The range is exponential, so increasing the CRF value +6 is roughly half the bitrate while -6 is roughly twice the bitrate. General usage is to choose the highest CRF value that still provides an acceptable quality. If the output looks good, then try a higher value and if it looks bad then choose a lower value.

> **Note:** The CRF quantizer scale mentioned on this page only applies to 8-bit x264 (10-bit x264 quantizer scale is 0-63). You can see what you are using with `x264 --help` listed under `Output bit depth`. 8-bit is more common among distributors.

2. Choose a preset

A preset is a collection of options that will provide a certain encoding speed to compression ratio. A slower preset will provide better compression (compression is quality per filesize). This means that, for example, if you target a certain file size or constant bit rate, you will achieve better quality with a slower preset. Similarly, for constant quality encoding, you will simply save bitrate by choosing a slower preset.

The general guideline is to use the slowest preset that you have patience for. Current presets in descending order of speed are: `ultrafast`, `superfast`, `veryfast`, `faster`, `fast`, `medium`, `slow`, `slower`, `veryslow`, `placebo`. The default preset is `medium`. Ignore `placebo` as it is not useful (see FAQ). You can see a list of current presets with `-preset help`, and what settings they apply with `x264 --fullhelp`.

You can optionally use `-tune` to change settings based upon the specifics of your input. Current tunings include: `film`, `animation`, `grain`, `stillimage`, `psnr`, `ssim`, `fastdecode`, `zerolatency`. For example, if your input is animation then use the `animation` tuning, or if you want to preserve grain then use the `grain` tuning. If you are unsure of what to use or your input does not match any of tunings then omit the `-tune` option. You can see a list of current tunings with `-tune help`, and what settings they apply with `x264 --fullhelp`.

Another optional setting is `-profile:v` which will limit the output to a specific H.264 profile. This can generally be omitted unless the target device only supports a certain profile (see Compatibility). Current profiles include: `baseline`, `main`, `high`, `high10`, `high422`, `high444`. Note that usage of `-profile:v` is incompatible with lossless encoding.

As a shortcut, you can also list all possible internal presets/tunes for FFmpeg by specifying no preset or tune option at all:

```
ffmpeg -y -i input -c:v libx264 -preset -tune -f mp4 /dev/null
```

> **Note:** Windows users should use `NUL` instead of `/dev/null`.

3. Use your settings

Once you've chosen your settings apply them for the rest of your videos if you are encoding more. This will ensure that they will all have similar quality.

CRF Example

```
ffmpeg -i input -c:v libx264 -preset slow -crf 22 -c:a copy output.mkv
```

Note that in this example the audio stream of the input file is simply stream copied over to the output and not re-encoded.

Two-Pass

This method is generally used if you are targeting a specific output file size and output quality from frame to frame is of less importance. This is best explained with an example. Your video is 10 minutes (600 seconds) long and an output of 50 MB is desired. Since `bitrate = file size / duration`:

```
(50 MB * 8192 [converts MB to kilobits]) / 600 seconds = ~683 kilobits/s total bitrate
683k - 128k (desired audio bitrate) = 555k video bitrate
```

Two-Pass Example

```
ffmpeg -y -i input -c:v libx264 -preset medium -b:v 555k -pass 1 -an -f mp4 /dev/null && \
ffmpeg -i input -c:v libx264 -preset medium -b:v 555k -pass 2 -c:a libfdk_aac -b:a 128k output.mp4
```

> **Note:** Windows users should use `NUL` instead of `/dev/null`.

As with CRF, choose the slowest preset you can tolerate.

Also see Making a high quality MPEG-4 ("DivX") rip of a DVD movie. It is an MEncoder guide, but it will give you an insight about how important it is to use two-pass when you want to efficiently use every bit when you're constrained with storage space.

Lossless H.264

You can use `-qp 0` or `-crf 0` to encode a lossless output. Use of `-qp` is recommended over `-crf` for lossless because 8-bit and 10-bit x264 use different `-crf` values for lossless. Two useful presets for this are `ultrafast` or `veryslow` since either a fast encoding speed or best compression are usually the most important factors. Most non-FFmpeg based players will not be able to decode lossless (but YouTube can), so if compatibility is an issue you should not use lossless.

Lossless Example (fastest encoding)

```
ffmpeg -i input -c:v libx264 -preset ultrafast -qp 0 output.mkv
```

Lossless Example (best compression)

```
ffmpeg -i input -c:v libx264 -preset veryslow -qp 0 output.mkv
```

Overwriting default preset settings

You can overwrite default preset settings with the `x264opts` option, the `x264-params` option, or by using the libx264 private options (see `ffmpeg -h encoder=libx264`). This is not recommended unless you know what you are doing. The presets were created by the x264 developers and tweaking values to get a better output is usually a waste of time.

Example:

```
ffmpeg -i input -c:v libx264 -preset slow -crf 22 -x264opts keyint=123:min-keyint=20 -c:a copy output.m
```

Additional Information & Tips

ABR (Average Bit Rate)

```
ffmpeg -i input -c:v libx264 -b:v 1000k output.mp4
```

This provides something of a "running average" target, with the end goal that the final file match this number "overall on average" (so basically, if it gets a lot of black frames, which cost very little, it will encode them with less than the requested bitrate, but then the next few seconds of (non-black) frames it will encode at very high quality, to bring the average back in line). Using 2-pass can help this method to be more effective. You can also use this in combination with a "max bit rate" setting in order to prevent some of the swings.

CBR (Constant Bit Rate)

There is no native CBR mode, but you can "simulate" a constant bit rate setting by tuning the parameters of ABR:

```
ffmpeg -i input -c:v libx264 -b:v 4000k -minrate 4000k -maxrate 4000k -bufsize 1835k out.m2v
```

In the above example, `-bufsize` is the "rate control buffer" so it will enforce your requested "average" (4000k in this case) across each 1835k worth of video. So basically it is assumed that the receiver/end player will buffer that much data so it's ok to fluctuate within that much.

Of course, if it's all just empty/black frames then it will still serve less than that many bits/s but it will raise the quality level as much as it can, up to the crf level.

CRF with maximum bit rate

You can also also use a crf with a maximum bit rate by specifying both crf *and* maxrate settings, like

```
ffmpeg -i input -c:v libx264 -crf 20 -maxrate 400k -bufsize 1835k output.mp4
```

This will effectively "target" crf 20, but if the output exceeds 400kb/s, it will degrade to something less than crf 20 in that case.

Low Latency

libx264 offers a `-tune zerolatency` option. See the StreamingGuide.

Compatibility

All devices

If you want your videos to have highest compatibility with target devices (older iOS versions or all Android devices):

```
-profile:v baseline -level 3.0
```

This disables some advanced features but provides for better compatibility. Typically you may not need this setting (and therefore avoid using `-profile:v` and `-level`), but if you do use this setting it may increase the bit rate quite a bit compared to what is needed to achieve the same quality in higher profiles.

iOS

iOS Compatability (source)			
Profile	**Level**	**Devices**	**Options**
Baseline	3.0	All devices	`-profile:v baseline -level 3.0`
Baseline	3.1	iPhone 3G and later, iPod touch 2nd generation and later	`-profile:v baseline -level 3.1`
Main	3.1	iPad (all versions), Apple TV 2 and later, iPhone 4 and later	`-profile:v main -level 3.1`
Main	4.0	Apple TV 3 and later, iPad 2 and later, iPhone 4S and later	`-profile:v main -level 4.0`
High	4.0	Apple TV 3 and later, iPad 2 and later, iPhone 4S and later	`-profile:v high -level 4.0`
High	4.1	iPad 2 and later and iPhone 4S and later	`-profile:v high -level 4.1`

This table does not include any additional restrictions which may be present.

Apple Quicktime

Apple QuickTime only supports YUV planar color space with 4:2:0 chroma subsampling (use `-pix_fmt yuv420p`) for H.264 video. For information on additional restrictions see QuickTime-compatible Encoding.

Blu-ray

See Authoring a professional Blu-ray Disc with x264.

Pre-testing your settings

Encode a random section instead of the whole video with the `-ss` and `-t` options to quickly get a general idea of what the output will look like.

- `-ss`: Offset time from beginning. Value can be in seconds or HH:MM:SS format.
- `-t`: Output duration. Value can be in seconds or HH:MM:SS format.

`faststart` for web video

You can add `-movflags +faststart` as an output option if your videos are going to be viewed in a browser. This will move some information to the beginning of your file and allow the video to begin playing before it is completely downloaded by the viewer. It is not required if you are going to use a video service such as YouTube.

FAQ

Will two-pass provide a better quality than CRF?

No, though it does allow you to target a file size more accurately.

Why is `placebo` a waste of time?

It helps at most ~1% compared to the `veryslow` preset at the cost of a much higher encoding time. It's diminishing returns: `veryslow` helps about 3% compared to the `slower` preset, `slower` helps about 5% compared to the `slow` preset, and `slow` helps about 5-10% compared to the `medium` preset.

Why doesn't my lossless output look lossless?

Blame the RGB to YUV color space conversion. If you convert to yuv444 it should still be lossless (which is the default now).

Will a graphics card make x264 encode faster?

Not necessarily. x264 supports OpenCL for some lookahead operations. There are also some proprietary encoders that utilize the GPU, but that does not mean they are well optimized, though encoding time may be faster; and they might be worse than vanilla x264, and possibly slower. Regardless, FFmpeg today doesn't support any means of GPU encoding, outside of libx264.

Encoding for dumb players

You may need to use `-pix_fmt yuv420p` for your output to work in QuickTime and most other players. These players only supports the YUV planar color space with 4:2:0 chroma subsampling for H.264 video. Otherwise, depending on your source, ffmpeg may output to a pixel format that may be incompatible with these players.

Additional Resources

- x264 Settings - MeWiki
- x264 Encoding Suggestions - MeWiki
- Constant Rate Factor Guide (copy of a deleted Handbrake Wiki page)

Last modified 3 weeks ago

Download in other formats:
Plain Text

Introduction

The best way to understand -map option is to think of it like a way to tell FFmpeg which streams do you want to select/copy from input to output.

The order of -map options, specified on cmd line, will create the same order of streams in the output file.

Here are several examples.

Input file

In all following examples, we will use an example input file like this one:

```
# fmpeg -i input.mkv

ffmpeg version ... Copyright (c) 2000-2012 the FFmpeg developers
...
Input #0, matroska,webm, from 'input.mkv':
  Duration: 01:39:44.02, start: 0.000000, bitrate: 5793 kb/s
    Stream #0:0(eng): Video: h264 (High), yuv420p, 1920x800, 23.98 fps, 23.98 tbr, 1k tbn, 47.95 tbc (c
    Stream #0:1(ger): Audio: dts (DTS), 48000 Hz, 5.1(side), s16, 1536 kb/s (default)
    Stream #0:2(eng): Audio: dts (DTS), 48000 Hz, 5.1(side), s16, 1536 kb/s
    Stream #0:3(ger): Subtitle: text (default)
At least one output file must be specified

#
```

Example 1

Now, let's say we wan't to:

- copy video stream
- encode german audio stream to mp3 (128kbps) and aac (96kbps) (creating 2 audio streams in the output)
- drop english audio stream
- copy subtitle stream

This can be done using the following FFmpeg command line:

```
ffmpeg -i input.mkv \
    -map 0:0 -map 0:1 -map 0:1 -map 0:3 \
    -c:v copy \
    -c:a:0 libmp3lame -b:a:0 128k \
    -c:a:1 libfaac -b:a:1 96k \
    -c:s copy \
    output.mkv
```

Note there is no "-map 0:2" and that "-map 0:1" has been specified twice.

Using "-map 0:0 -map 0:1 -map 0:1 -map 0:3" we told FFmpeg to select/map specified input streams to output in that order.

So, our output will now have the following streams:

```
Output #0, matroska, to 'output.mkv':
    Stream #0:0(eng): Video ...
    Stream #0:1(ger): Audio ...
    Stream #0:2(ger): Audio ...
    Stream #0:3(ger): Subtitle ...
```

After we selected which streams we would like in our output, using "-map" option, we specified codecs for each stream in our output.

Video and subtitle stream have just been copied and german audio stream has been encoded to 2 new audio streams, mp3 and aac.

We used "-c:a:0" to specify codec for the output's first AUDIO stream and "-c:a:1" to specify codec for the output's second AUDIO stream.

Note that "a:0" refers to the output's first AUDIO stream (#0:1 in our case), "a:1" refers to the output's 2nd AUDIO stream (#0:2 in our case), etc.

The result will be:

```
Output #0, matroska, to 'output.mkv':
    Stream #0:0(eng): Video ...
    Stream #0:1(ger): Audio ...
    Stream #0:2(ger): Audio ...
    Stream #0:3(ger): Subtitle ...
Stream mapping:
  Stream #0:0 -> #0:0 (copy)
  Stream #0:1 -> #0:1 (dca -> libmp3lame)
  Stream #0:2 -> #0:2 (dca -> libfaac)
  Stream #0:3 -> #0:3 (copy)
```

Example 2

Let's say that we want to reorder input streams backwards, so that we have output like this:

```
    Stream #0:0(ger): Subtitle: text (default)
    Stream #0:1(eng): Audio: dts (DTS), 48000 Hz, 5.1(side), s16, 1536 kb/s
    Stream #0:2(ger): Audio: dts (DTS), 48000 Hz, 5.1(side), s16, 1536 kb/s (default)
    Stream #0:3(eng): Video: h264 (High), yuv420p, 1920x800, 23.98 fps, 23.98 tbr, 1k tbn, 47.95 tbc (c
```

This can simply be done using the following command line:

```
ffmpeg -i input.mkv -map 0:3 -map 0:2 -map 0:1 -map 0:0 -c copy output.mkv
```

Note that we specified all the input streams, but in the reverse order, which causes that order to be respected in the output.

The option "-c copy" tells FFmpeg to use "copy" on all streams.

Example 3

If we want to extract only audio streams, from input file, then we can do it like this:

```
ffmpeg -i input.mkv -map 0:1 -map 0:2 -c copy output.mkv
```

Example 4

If we want to re-encode just the video streams, but copy all the other streams (like audio, subtitles, attachments, etc), we might use something like this:

```
ffmpeg -i input.mkv -map 0 -c copy -c:v mpeg2video output.mkv
```

It will tell ffmpeg to:

- read the input file 'input.mkv'
- select all the input streams (first input = 0) to be processed (using "-map 0")
- mark all the streams to be just copied to the output (using "-c copy")
- mark just the video streams to be re-encoded (using "-c:v mpeg2video")
- write the output file 'output.mkv'

ffmpeg Documentation

Table of Contents

1. Synopsis

ffmpeg [*global_options*] {[*input_file_options*] -i ‘`input_file`’} ... {[*output_file_options*] ‘`output_file`’} ...

2. Description

`ffmpeg` is a very fast video and audio converter that can also grab from a live audio/video source. It can also convert between arbitrary sample rates and resize video on the fly with a high quality polyphase filter.

`ffmpeg` reads from an arbitrary number of input "files" (which can be regular files, pipes, network streams, grabbing devices, etc.), specified by the `-i` option, and writes to an arbitrary number of output "files", which are specified by a plain output filename. Anything found on the command line which cannot be interpreted as an option is considered to be an output filename.

Each input or output file can, in principle, contain any number of streams of different types (video/audio/subtitle/attachment/data). The allowed number and/or types of streams may be limited by the container format. Selecting which streams from which inputs will go into which output is either done automatically or with the `-map` option (see the Stream selection chapter).

To refer to input files in options, you must use their indices (0-based). E.g. the first input file is `0`, the second is `1`, etc. Similarly, streams within a file are referred to by their indices. E.g. `2:3` refers to the fourth stream in the third input file. Also see the Stream specifiers chapter.

As a general rule, options are applied to the next specified file. Therefore, order is important, and you can have the same option on the command line multiple times. Each occurrence is then applied to the next input or output file. Exceptions from this rule are the global options (e.g. verbosity level), which should be specified first.

Do not mix input and output files – first specify all input files, then all output files. Also do not mix options which belong to different files. All options apply ONLY to the next input or output file and are reset between files.

- To set the video bitrate of the output file to 64 kbit/s:

```
ffmpeg -i input.avi -b:v 64k -bufsize 64k output.avi
```

- To force the frame rate of the output file to 24 fps:

```
ffmpeg -i input.avi -r 24 output.avi
```

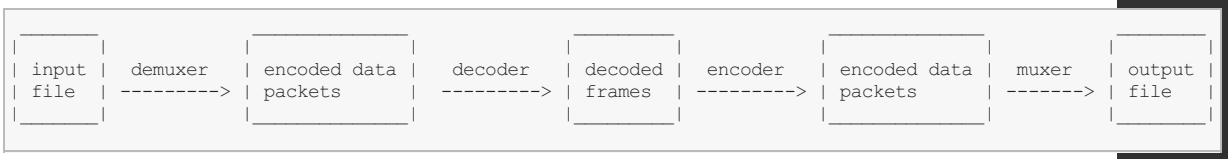
- To force the frame rate of the input file (valid for raw formats only) to 1 fps and the frame rate of the output file to 24 fps:

```
ffmpeg -r 1 -i input.m2v -r 24 output.avi
```

The format option may be needed for raw input files.

3. Detailed description

The transcoding process in `ffmpeg` for each output can be described by the following diagram:



`ffmpeg` calls the libavformat library (containing demuxers) to read input files and get packets containing encoded data from them. When there are multiple input files, `ffmpeg` tries to keep them synchronized by tracking lowest timestamp on any active input stream.

Encoded packets are then passed to the decoder (unless streamcopy is selected for the stream, see further for a description). The decoder produces uncompressed frames (raw video/PCM audio/...) which can be processed further by filtering (see next section). After filtering, the frames are passed to the encoder, which encodes them and outputs encoded packets. Finally those are passed to the muxer, which writes the encoded packets to the output file.

3.1 Filtering

Before encoding, `ffmpeg` can process raw audio and video frames using filters from the libavfilter library. Several chained filters form a filter graph. `ffmpeg` distinguishes between two types of filtergraphs: simple and complex.

3.1.1 Simple filtergraphs

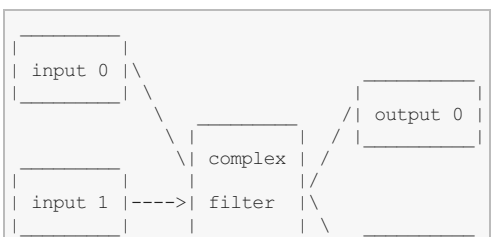
Simple filtergraphs are those that have exactly one input and output, both of the same type. In the above diagram they can be represented by simply inserting an additional step between decoding and encoding:

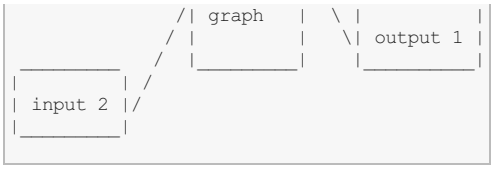
Simple filtergraphs are configured with the per-stream '`-filter`' option (with '`-vf`' and '`-af`' aliases for video and audio respectively). A simple filtergraph for video can look for example like this:

Note that some filters change frame properties but not frame contents. E.g. the `fps` filter in the example above changes number of frames, but does not touch the frame contents. Another example is the `setpts` filter, which only sets timestamps and otherwise passes the frames unchanged.

3.1.2 Complex filtergraphs

Complex filtergraphs are those which cannot be described as simply a linear processing chain applied to one stream. This is the case, for example, when the graph has more than one input and/or output, or when output stream type is different from input. They can be represented with the following diagram:





Complex filtergraphs are configured with the '`-filter_complex`' option. Note that this option is global, since a complex filtergraph, by its nature, cannot be unambiguously associated with a single stream or file.

The '`-lavfi`' option is equivalent to '`-filter_complex`'.

A trivial example of a complex filtergraph is the `overlay` filter, which has two video inputs and one video output, containing one video overlaid on top of the other. Its audio counterpart is the `amix` filter.

3.2 Stream copy

Stream copy is a mode selected by supplying the `copy` parameter to the '`-codec`' option. It makes `ffmpeg` omit the decoding and encoding step for the specified stream, so it does only demuxing and muxing. It is useful for changing the container format or modifying container-level metadata. The diagram above will, in this case, simplify to this:

Since there is no decoding or encoding, it is very fast and there is no quality loss. However, it might not work in some cases because of many factors. Applying filters is obviously also impossible, since filters work on uncompressed data.

4. Stream selection

By default, `ffmpeg` includes only one stream of each type (video, audio, subtitle) present in the input files and adds them to each output file. It picks the "best" of each based upon the following criteria: for video, it is the stream with the highest resolution, for audio, it is the stream with the most channels, for subtitles, it is the first subtitle stream. In the case where several streams of the same type rate equally, the stream with the lowest index is chosen.

You can disable some of those defaults by using the `-vn/-an/-sn` options. For full manual control, use the `-map` option, which disables the defaults just described.

5. Options

All the numerical options, if not specified otherwise, accept a string representing a number as input, which may be followed by one of the SI unit prefixes, for example: 'K', 'M', or 'G'.

If 'i' is appended to the SI unit prefix, the complete prefix will be interpreted as a unit prefix for binary multiples, which are based on powers of 1024 instead of powers of 1000. Appending 'B' to the SI unit prefix multiplies the value by 8. This allows using, for example: 'KB', 'MiB', 'G' and 'B' as number suffixes.

Options which do not take arguments are boolean options, and set the corresponding value to true. They can be set to false by prefixing the option name with "no". For example using "-nofoo" will set the boolean option with name "foo" to false.

5.1 Stream specifiers

Some options are applied per-stream, e.g. bitrate or codec. Stream specifiers are used to precisely specify which stream(s) a given option belongs to.

A stream specifier is a string generally appended to the option name and separated from it by a colon. E.g. `-codec:a:1 ac3` contains the `a:1` stream specifier, which matches the second audio stream. Therefore, it would select the ac3 codec for the second audio stream.

A stream specifier can match several streams, so that the option is applied to all of them. E.g. the stream specifier in `-b:a 128k` matches all audio streams.

An empty stream specifier matches all streams. For example, `-codec copy` or `-codec: copy` would copy all the streams without reencoding.

Possible forms of stream specifiers are:

'***stream_index***'

Matches the stream with this index. E.g. `-threads:1 4` would set the thread count for the second stream to 4.

'***stream_type*[:*stream_index*]**'

stream_type is one of following: 'v' for video, 'a' for audio, 's' for subtitle, 'd' for data, and 't' for attachments. If *stream_index* is given, then it matches stream number *stream_index* of this type. Otherwise, it matches all streams of this type.

'**p:*program_id*[:*stream_index*]**'

If *stream_index* is given, then it matches the stream with number *stream_index* in the program with the id *program_id*. Otherwise, it matches all streams in the program.

'**#*stream_id***'

Matches the stream by a format-specific ID.

5.2 Generic options

These options are shared amongst the ff* tools.

'**-L**'

Show license.

'**-h, -?, -help, --help [*arg*]**'

Show help. An optional parameter may be specified to print help about a specific item. If no argument is specified, only basic (non advanced) tool options are shown.

Possible values of *arg* are:

'**long**'

Print advanced tool options in addition to the basic tool options.

'**full**'

Print complete list of options, including shared and private options for encoders, decoders, demuxers, muxers, filters, etc.

'**decoder=*decoder_name***'

Print detailed information about the decoder named *decoder_name*. Use the '`-decoders`' option to get a list of all decoders.

'**encoder=*encoder_name***'

Print detailed information about the encoder named *encoder_name*. Use the '`-encoders`' option to get a list of all encoders.

'**demuxer=*demuxer_name***'

Print detailed information about the demuxer named *demuxer_name*. Use the '`-formats`' option to get a list of all demuxers and muxers.

'**muxer=*muxer_name***'

Print detailed information about the muxer named *muxer_name*. Use the '`-formats`' option to get a list of all muxers and demuxers.

'**filter=*filter_name***'

Print detailed information about the filter name *filter_name*. Use the '`-filters`' option to get a list of all filters.

'**-version**'

Show version.

'**-formats**'

Show available formats.

'**-codecs**'

Show all codecs known to libavcodec.

Note that the term 'codec' is used throughout this documentation as a shortcut for what is more correctly called a media bitstream format.

'**-decoders**'

Show available decoders.

'**-encoders**'

Show all available encoders.

‘**-bsfs**’

Show available bitstream filters.

‘**-protocols**’

Show available protocols.

‘**-filters**’

Show available libavfilter filters.

‘**-pix_fmts**’

Show available pixel formats.

‘**-sample_fmts**’

Show available sample formats.

‘**-layouts**’

Show channel names and standard channel layouts.

‘**-colors**’

Show recognized color names.

‘**-loglevel [repeat+]*loglevel* | -v [repeat+]*loglevel***’

Set the logging level used by the library. Adding "repeat+" indicates that repeated log output should not be compressed to the first line and the "Last message repeated n times" line will be omitted. "repeat" can also be used alone. If "repeat" is used alone, and with no prior loglevel set, the default loglevel will be used. If multiple loglevel parameters are given, using ’repeat’ will not change the loglevel. *loglevel* is a number or a string containing one of the following values:

‘**quiet**’

Show nothing at all; be silent.

‘**panic**’

Only show fatal errors which could lead the process to crash, such as and assert failure. This is not currently used for anything.

‘**fatal**’

Only show fatal errors. These are errors after which the process absolutely cannot continue after.

‘**error**’

Show all errors, including ones which can be recovered from.

‘**warning**’

Show all warnings and errors. Any message related to possibly incorrect or unexpected events will be shown.

‘**info**’

Show informative messages during processing. This is in addition to warnings and errors. This is the default value.

‘**verbose**’

Same as `info`, except more verbose.

‘**debug**’

Show everything, including debugging information.

By default the program logs to stderr, if coloring is supported by the terminal, colors are used to mark errors and warnings. Log coloring can be disabled setting the environment variable `AV_LOG_FORCE_NOCOLOR` or `NO_COLOR`, or can be forced setting the environment variable `AV_LOG_FORCE_COLOR`. The use of the environment variable `NO_COLOR` is deprecated and will be dropped in a following FFmpeg version.

‘**-report**’

Dump full command line and console output to a file named *program*`-`*YYYYMMDD*`-`*HHMMSS*`.log` in the current directory. This file can be useful for bug reports. It also implies `-loglevel verbose`.

Setting the environment variable `FFREPORT` to any value has the same effect. If the value is a ’:’-separated key=value sequence, these options will affect the report; options values must be escaped if they contain special characters or the options delimiter ’:’ (see the “Quoting and escaping” section in the ffmpeg-utils manual). The following option is

recognized:

'file'

set the file name to use for the report; `%p` is expanded to the name of the program, `%t` is expanded to a timestamp, `%%` is expanded to a plain `%`

Errors in parsing the environment variable are not fatal, and will not appear in the report.

'-hide_banner'

Suppress printing banner.

All FFmpeg tools will normally show a copyright notice, build options and library versions. This option can be used to suppress printing this information.

'-cpuflags flags (*global*)'

Allows setting and clearing cpu flags. This option is intended for testing. Do not use it unless you know what you're doing.

```
ffmpeg -cpuflags -sse+mmx ...
ffmpeg -cpuflags mmx ...
ffmpeg -cpuflags 0 ...
```

Possible flags for this option are:

'x86'

'mmx'
'mmxext'
'sse'
'sse2'
'sse2slow'
'sse3'
'sse3slow'
'ssse3'
'atom'
'sse4.1'
'sse4.2'
'avx'
'xop'
'fma4'
'3dnow'
'3dnowext'
'cmov'

'ARM'

'armv5te'
'armv6'
'armv6t2'
'vfp'
'vfpv3'
'neon'

'PowerPC'

'altivec'

'Specific Processors'

'pentium2'
'pentium3'
'pentium4'
'k6'
'k62'
'athlon'
'athlonxp'
'k8'

'-opencl_bench'

Benchmark all available OpenCL devices and show the results. This option is only available when FFmpeg has been compiled with `--enable-opencl`.

'-opencl_options options (*global*)'

Set OpenCL environment options. This option is only available when FFmpeg has been compiled with `--enable-`

`opencl`.

options must be a list of *key=value* option pairs separated by ':'. See the "OpenCL Options" section in the ffmpeg-utils manual for the list of supported options.

5.3 AVOptions

These options are provided directly by the libavformat, libavdevice and libavcodec libraries. To see the list of available AVOptions, use the '`-help`' option. They are separated into two categories:

'`generic`'

These options can be set for any container, codec or device. Generic options are listed under AVFormatContext options for containers/devices and under AVCodecContext options for codecs.

'`private`'

These options are specific to the given container, device or codec. Private options are listed under their corresponding containers/devices/codecs.

For example to write an ID3v2.3 header instead of a default ID3v2.4 to an MP3 file, use the '`id3v2_version`' private option of the MP3 muxer:

```
ffmpeg -i input.flac -id3v2_version 3 out.mp3
```

All codec AVOptions are per-stream, and thus a stream specifier should be attached to them.

Note: the '`-nooption`' syntax cannot be used for boolean AVOptions, use '`-option 0`'/'`-option 1`'.

Note: the old undocumented way of specifying per-stream AVOptions by prepending v/a/s to the options name is now obsolete and will be removed soon.

5.4 Main options

'`-f` *`fmt (input/output)`*'

Force input or output file format. The format is normally auto detected for input files and guessed from the file extension for output files, so this option is not needed in most cases.

'`-i` *`filename (input)`*'

input file name

'`-y` *`(global)`*'

Overwrite output files without asking.

'`-n` *`(global)`*'

Do not overwrite output files, and exit immediately if a specified output file already exists.

'`-c[:`*`stream_specifier`*`]` *`codec (input/output,per-stream)`*'
'`-codec[:`*`stream_specifier`*`]` *`codec (input/output,per-stream)`*'

Select an encoder (when used before an output file) or a decoder (when used before an input file) for one or more streams. *codec* is the name of a decoder/encoder or a special value `copy` (output only) to indicate that the stream is not to be re-encoded.

For example

```
ffmpeg -i INPUT -map 0 -c:v libx264 -c:a copy OUTPUT
```

encodes all video streams with libx264 and copies all audio streams.

For each stream, the last matching `c` option is applied, so

```
ffmpeg -i INPUT -map 0 -c copy -c:v:1 libx264 -c:a:137 libvorbis OUTPUT
```

will copy all the streams except the second video, which will be encoded with libx264, and the 138th audio, which will be encoded with libvorbis.

'`-t` *`duration (output)`*'

Stop writing the output after its duration reaches *duration*. *duration* may be a number in seconds, or in `hh:mm:ss[.xxx]` form.

-to and -t are mutually exclusive and -t has priority.

'-to *position* (*output*)'

Stop writing the output at *position*. *position* may be a number in seconds, or in `hh:mm:ss[.xxx]` form.

-to and -t are mutually exclusive and -t has priority.

'-fs *limit_size* (*output*)'

Set the file size limit, expressed in bytes.

'-ss *position* (*input/output*)'

When used as an input option (before `-i`), seeks in this input file to *position*. Note the in most formats it is not possible to seek exactly, so `ffmpeg` will seek to the closest seek point before *position*. When transcoding and '`-accurate_seek`' is enabled (the default), this extra segment between the seek point and *position* will be decoded and discarded. When doing stream copy or when '`-noaccurate_seek`' is used, it will be preserved.

When used as an output option (before an output filename), decodes but discards input until the timestamps reach *position*.

position may be either in seconds or in `hh:mm:ss[.xxx]` form.

'-itsoffset *offset* (*input*)'

Set the input time offset in seconds. `[-]hh:mm:ss[.xxx]` syntax is also supported. The offset is added to the timestamps of the input files. Specifying a positive offset means that the corresponding streams are delayed by *offset* seconds.

'-timestamp *time* (*output*)'

Set the recording timestamp in the container. The syntax for *time* is:

```
now|([(YYYY-MM-DD|YYYYMMDD)[T|t| ]]((HH:MM:SS[.m...])|(HHMMSS[.m...]))[Z|z])
```

If the value is "now" it takes the current time. Time is local time unless 'Z' or 'z' is appended, in which case it is interpreted as UTC. If the year-month-day part is not specified it takes the current year-month-day.

'-metadata[:metadata_specifier] *key=value* (*output,per-metadata*)'

Set a metadata key/value pair.

An optional *metadata_specifier* may be given to set metadata on streams or chapters. See `-map_metadata` documentation for details.

This option overrides metadata set with `-map_metadata`. It is also possible to delete metadata by using an empty value.

For example, for setting the title in the output file:

```
ffmpeg -i in.avi -metadata title="my title" out.flv
```

To set the language of the first audio stream:

```
ffmpeg -i INPUT -metadata:s:a:1 language=eng OUTPUT
```

'-target *type* (*output*)'

Specify target file type (`vcd`, `svcd`, `dvd`, `dv`, `dv50`). *type* may be prefixed with `pal-`, `ntsc-` or `film-` to use the corresponding standard. All the format options (bitrate, codecs, buffer sizes) are then set automatically. You can just type:

```
ffmpeg -i myfile.avi -target vcd /tmp/vcd.mpg
```

Nevertheless you can specify additional options as long as you know they do not conflict with the standard, as in:

```
ffmpeg -i myfile.avi -target vcd -bf 2 /tmp/vcd.mpg
```

'-dframes *number* (*output*)'

Set the number of data frames to record. This is an alias for `-frames:d`.

'-frames[:*stream_specifier*] *framecount* (*output,per-stream*)'

Stop writing to the stream after *framecount* frames.

'-q[:*stream_specifier*] *q* (*output,per-stream*)'
'-qscale[:*stream_specifier*] *q* (*output,per-stream*)'

Use fixed quality scale (VBR). The meaning of *q*/*qscale* is codec-dependent. If *qscale* is used without a *stream_specifier* then it applies only to the video stream, this is to maintain compatibility with previous behavior and as specifying the same codec specific value to 2 different codecs that is audio and video generally is not what is intended when no stream_specifier is used.

'-filter[:*stream_specifier*] *filtergraph* (*output,per-stream*)'

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

filtergraph is a description of the filtergraph to apply to the stream, and must have a single input and a single output of the same type of the stream. In the filtergraph, the input is associated to the label `in`, and the output to the label `out`. See the ffmpeg-filters manual for more information about the filtergraph syntax.

See the -filter_complex option if you want to create filtergraphs with multiple inputs and/or outputs.

'-filter_script[:*stream_specifier*] *filename* (*output,per-stream*)'

This option is similar to '`-filter`', the only difference is that its argument is the name of the file from which a filtergraph description is to be read.

'-pre[:*stream_specifier*] *preset_name* (*output,per-stream*)'

Specify the preset for matching stream(s).

'-stats (*global*)'

Print encoding progress/statistics. It is on by default, to explicitly disable it you need to specify `-nostats`.

'-progress *url* (*global*)'

Send program-friendly progress information to *url*.

Progress information is written approximately every second and at the end of the encoding process. It is made of "*key*=*value*" lines. *key* consists of only alphanumeric characters. The last key of a sequence of progress information is always "progress".

'-stdin'

Enable interaction on standard input. On by default unless standard input is used as an input. To explicitly disable interaction you need to specify `-nostdin`.

Disabling interaction on standard input is useful, for example, if ffmpeg is in the background process group. Roughly the same result can be achieved with `ffmpeg ... < /dev/null` but it requires a shell.

'-debug_ts (*global*)'

Print timestamp information. It is off by default. This option is mostly useful for testing and debugging purposes, and the output format may change from one version to another, so it should not be employed by portable scripts.

See also the option `-fdebug ts`.

'-attach *filename* (*output*)'

Add an attachment to the output file. This is supported by a few formats like Matroska for e.g. fonts used in rendering subtitles. Attachments are implemented as a specific type of stream, so this option will add a new stream to the file. It is then possible to use per-stream options on this stream in the usual way. Attachment streams created with this option will be created after all the other streams (i.e. those created with `-map` or automatic mappings).

Note that for Matroska you also have to set the mimetype metadata tag:

```
ffmpeg -i INPUT -attach DejaVuSans.ttf -metadata:s:2 mimetype=application/x-truetype-font out.mkv
```

(assuming that the attachment stream will be third in the output file).

'-dump_attachment[:*stream_specifier*] *filename* (*input,per-stream*)'

Extract the matching attachment stream into a file named *filename*. If *filename* is empty, then the value of the `filename` metadata tag will be used.

E.g. to extract the first attachment to a file named 'out.ttf':

```
ffmpeg -dump_attachment:t:0 out.ttf -i INPUT
```

To extract all attachments to files determined by the `filename` tag:

```
ffmpeg -dump_attachment:t "" -i INPUT
```

Technical note – attachments are implemented as codec extradata, so this option can actually be used to extract extradata from any stream, not just attachments.

5.5 Video Options

‘-vframes *number* (*output*)’

Set the number of video frames to record. This is an alias for `-frames:v`.

‘-r[:*stream_specifier*] *fps* (*input/output,per-stream*)’

Set frame rate (Hz value, fraction or abbreviation).

As an input option, ignore any timestamps stored in the file and instead generate timestamps assuming constant frame rate *fps*.

As an output option, duplicate or drop input frames to achieve constant output frame rate *fps*.

‘-s[:*stream_specifier*] *size* (*input/output,per-stream*)’

Set frame size.

As an input option, this is a shortcut for the ‘`video_size`’ private option, recognized by some demuxers for which the frame size is either not stored in the file or is configurable – e.g. raw video or video grabbers.

As an output option, this inserts the `scale` video filter to the *end* of the corresponding filtergraph. Please use the `scale` filter directly to insert it at the beginning or some other place.

The format is ‘`wxh`’ (default - same as source).

‘-aspect[:*stream_specifier*] *aspect* (*output,per-stream*)’

Set the video display aspect ratio specified by *aspect*.

aspect can be a floating point number string, or a string of the form *num*:*den*, where *num* and *den* are the numerator and denominator of the aspect ratio. For example "4:3", "16:9", "1.3333", and "1.7777" are valid argument values.

If used together with ‘`-vcodec copy`’, it will affect the aspect ratio stored at container level, but not the aspect ratio stored in encoded frames, if it exists.

‘-vn (*output*)’

Disable video recording.

‘-vcodec *codec* (*output*)’

Set the video codec. This is an alias for `-codec:v`.

‘-pass[:*stream_specifier*] *n* (*output,per-stream*)’

Select the pass number (1 or 2). It is used to do two-pass video encoding. The statistics of the video are recorded in the first pass into a log file (see also the option -passlogfile), and in the second pass that log file is used to generate the video at the exact requested bitrate. On pass 1, you may just deactivate audio and set output to null, examples for Windows and Unix:

```
ffmpeg -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y NUL
ffmpeg -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y /dev/null
```

‘-passlogfile[:*stream_specifier*] *prefix* (*output,per-stream*)’

Set two-pass log file name prefix to *prefix*, the default file name prefix is “ffmpeg2pass”. The complete file name will be ‘`PREFIX-N.log`’, where N is a number specific to the output stream

‘-vf *filtergraph* (*output*)’

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

This is an alias for `-filter:v`, see the -filter option.

5.6 Advanced Video Options

‘-pix_fmt[:*stream_specifier*] *format* (*input/output,per-stream*)’

Set pixel format. Use `-pix_fmts` to show all the supported pixel formats. If the selected pixel format can not be selected, ffmpeg will print a warning and select the best pixel format supported by the encoder. If *pix_fmt* is prefixed by a `+`, ffmpeg will exit with an error if the requested pixel format can not be selected, and automatic conversions

inside filtergraphs are disabled. If *pix_fmt* is a single `+`, ffmpeg selects the same pixel format as the input (or graph output) and automatic conversions are disabled.

'-sws_flags *flags* (*input/output*)'

Set SwScaler flags.

'-vdt *n*'

Discard threshold.

'-rc_override[:*stream_specifier*] *override* (*output,per-stream*)'

Rate control override for specific intervals, formatted as "int,int,int" list separated with slashes. Two first values are the beginning and end frame numbers, last one is quantizer to use if positive, or quality factor if negative.

'-ilme'

Force interlacing support in encoder (MPEG-2 and MPEG-4 only). Use this option if your input file is interlaced and you want to keep the interlaced format for minimum losses. The alternative is to deinterlace the input stream with '`-deinterlace`', but deinterlacing introduces losses.

'-psnr'

Calculate PSNR of compressed frames.

'-vstats'

Dump video coding statistics to '`vstats_HHMMSS.log`'.

'-vstats_file *file*'

Dump video coding statistics to *file*.

'-top[:*stream_specifier*] *n* (*output,per-stream*)'

top=1/bottom=0/auto=-1 field first

'-dc *precision*'

Intra_dc_precision.

'-vtag *fourcc/tag* (*output*)'

Force video tag/fourcc. This is an alias for `-tag:v`.

'-qphist (*global*)'

Show QP histogram

'-vbsf *bitstream_filter*'

Deprecated see -bsf

'-force_key_frames[:*stream_specifier*] *time*[,*time*...] (*output,per-stream*)'
'-force_key_frames[:*stream_specifier*] expr:*expr* (*output,per-stream*)'

Force key frames at the specified timestamps, more precisely at the first frames after each specified time.

If the argument is prefixed with `expr:`, the string *expr* is interpreted like an expression and is evaluated for each frame. A key frame is forced in case the evaluation is non-zero.

If one of the times is "`chapters`[*delta*]", it is expanded into the time of the beginning of all chapters in the file, shifted by *delta*, expressed as a time in seconds. This option can be useful to ensure that a seek point is present at a chapter mark or any other designated place in the output file.

For example, to insert a key frame at 5 minutes, plus key frames 0.1 second before the beginning of every chapter:

```
-force_key_frames 0:05:00,chapters-0.1
```

The expression in *expr* can contain the following constants:

'n'

the number of current processed frame, starting from 0

'n_forced'

the number of forced frames

'prev_forced_n'

the number of the previous forced frame, it is NAN when no keyframe was forced yet

'prev_forced_t'

the time of the previous forced frame, it is NAN when no keyframe was forced yet

't'

the time of the current processed frame

For example to force a key frame every 5 seconds, you can specify:

```
-force_key_frames expr:gte(t,n_forced*5)
```

To force a key frame 5 seconds after the time of the last forced one, starting from second 13:

```
-force_key_frames expr:if(isnan(prev_forced_t),gte(t,13),gte(t,prev_forced_t+5))
```

Note that forcing too many keyframes is very harmful for the lookahead algorithms of certain encoders: using fixed-GOP options or similar would be more efficient.

'**-copyinkf[:*stream_specifier*] (*output,per-stream*)**'

When doing stream copy, copy also non-key frames found at the beginning.

'**-hwaccel[:*stream_specifier*] *hwaccel* (*input,per-stream*)**'

Use hardware acceleration to decode the matching stream(s). The allowed values of *hwaccel* are:

'**none**'

Do not use any hardware acceleration (the default).

'**auto**'

Automatically select the hardware acceleration method.

'**vdpau**'

Use VDPAU (Video Decode and Presentation API for Unix) hardware acceleration.

This option has no effect if the selected hwaccel is not available or not supported by the chosen decoder.

Note that most acceleration methods are intended for playback and will not be faster than software decoding on modern CPUs. Additionally, ffmpeg will usually need to copy the decoded frames from the GPU memory into the system memory, resulting in further performance loss. This option is thus mainly useful for testing.

'**-hwaccel_device[:*stream_specifier*] *hwaccel_device* (*input,per-stream*)**'

Select a device to use for hardware acceleration.

This option only makes sense when the '-hwaccel' option is also specified. Its exact meaning depends on the specific hardware acceleration method chosen.

'**vdpau**'

For VDPAU, this option specifies the X11 display/screen to use. If this option is not specified, the value of the *DISPLAY* environment variable is used

5.7 Audio Options

'**-aframes *number* (*output*)**'

Set the number of audio frames to record. This is an alias for -frames:a.

'**-ar[:*stream_specifier*] *freq* (*input/output,per-stream*)**'

Set the audio sampling frequency. For output streams it is set by default to the frequency of the corresponding input stream. For input streams this option only makes sense for audio grabbing devices and raw demuxers and is mapped to the corresponding demuxer options.

'**-aq *q* (*output*)**'

Set the audio quality (codec-specific, VBR). This is an alias for -q:a.

'**-ac[:*stream_specifier*] *channels* (*input/output,per-stream*)**'

Set the number of audio channels. For output streams it is set by default to the number of input audio channels. For input streams this option only makes sense for audio grabbing devices and raw demuxers and is mapped to the

corresponding demuxer options.

'-an (*output*)'

Disable audio recording.

'-acodec *codec* (*input/output*)'

Set the audio codec. This is an alias for `-codec:a`.

'-sample_fmt[:*stream_specifier*] *sample_fmt* (*output,per-stream*)'

Set the audio sample format. Use `-sample_fmts` to get a list of supported sample formats.

'-af *filtergraph* (*output*)'

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

This is an alias for `-filter:a`, see the -filter option.

5.8 Advanced Audio options:

'-atag *fourcc/tag* (*output*)'

Force audio tag/fourcc. This is an alias for `-tag:a`.

'-absf *bitstream_filter*'

Deprecated, see -bsf

'-guess_layout_max *channels* (*input,per-stream*)'

If some input channel layout is not known, try to guess only if it corresponds to at most the specified number of channels. For example, 2 tells to `ffmpeg` to recognize 1 channel as mono and 2 channels as stereo but not 6 channels as 5.1. The default is to always try to guess. Use 0 to disable all guessing.

5.9 Subtitle options:

'-scodec *codec* (*input/output*)'

Set the subtitle codec. This is an alias for `-codec:s`.

'-sn (*output*)'

Disable subtitle recording.

'-sbsf *bitstream_filter*'

Deprecated, see -bsf

5.10 Advanced Subtitle options:

'-fix_sub_duration'

Fix subtitles durations. For each subtitle, wait for the next packet in the same stream and adjust the duration of the first to avoid overlap. This is necessary with some subtitles codecs, especially DVB subtitles, because the duration in the original packet is only a rough estimate and the end is actually marked by an empty subtitle frame. Failing to use this option when necessary can result in exaggerated durations or muxing failures due to non-monotonic timestamps.

Note that this option will delay the output of all data until the next subtitle packet is decoded: it may increase memory consumption and latency a lot.

'-canvas_size *size*'

Set the size of the canvas used to render subtitles.

5.11 Advanced options

'-map [-]*input_file_id*[:*stream_specifier*][,*sync_file_id*[:*stream_specifier*]] | *[linklabel]* (*output*)'

Designate one or more input streams as a source for the output file. Each input stream is identified by the input file index *input_file_id* and the input stream index *input_stream_id* within the input file. Both indices start at 0. If specified, *sync_file_id*:*stream_specifier* sets which input stream is used as a presentation sync reference.

The first `-map` option on the command line specifies the source for output stream 0, the second `-map` option specifies the source for output stream 1, etc.

A `-` character before the stream identifier creates a "negative" mapping. It disables matching streams from already created mappings.

An alternative *[linklabel]* form will map outputs from complex filter graphs (see the '`-filter_complex`' option) to the output file. *linklabel* must correspond to a defined output link label in the graph.

For example, to map ALL streams from the first input file to output

```
ffmpeg -i INPUT -map 0 output
```

For example, if you have two audio streams in the first input file, these streams are identified by "0:0" and "0:1". You can use `-map` to select which streams to place in an output file. For example:

```
ffmpeg -i INPUT -map 0:1 out.wav
```

will map the input stream in '`INPUT`' identified by "0:1" to the (single) output stream in '`out.wav`'.

For example, to select the stream with index 2 from input file '`a.mov`' (specified by the identifier "0:2"), and stream with index 6 from input '`b.mov`' (specified by the identifier "1:6"), and copy them to the output file '`out.mov`':

```
ffmpeg -i a.mov -i b.mov -c copy -map 0:2 -map 1:6 out.mov
```

To select all video and the third audio stream from an input file:

```
ffmpeg -i INPUT -map 0:v -map 0:a:2 OUTPUT
```

To map all the streams except the second audio, use negative mappings

```
ffmpeg -i INPUT -map 0 -map -0:a:1 OUTPUT
```

Note that using this option disables the default mappings for this output file.

'**`-map_channel [`*`input_file_id.stream_specifier.channel_id`*`|-1][:`*`output_file_id.stream_specifier`*`]`**'

Map an audio channel from a given input to an output. If *output_file_id.stream_specifier* is not set, the audio channel will be mapped on all the audio streams.

Using "-1" instead of *input_file_id.stream_specifier.channel_id* will map a muted channel.

For example, assuming *INPUT* is a stereo audio file, you can switch the two audio channels with the following command:

```
ffmpeg -i INPUT -map_channel 0.0.1 -map_channel 0.0.0 OUTPUT
```

If you want to mute the first channel and keep the second:

```
ffmpeg -i INPUT -map_channel -1 -map_channel 0.0.1 OUTPUT
```

The order of the "-map_channel" option specifies the order of the channels in the output stream. The output channel layout is guessed from the number of channels mapped (mono if one "-map_channel", stereo if two, etc.). Using "-ac" in combination of "-map_channel" makes the channel gain levels to be updated if input and output channel layouts don't match (for instance two "-map_channel" options and "-ac 6").

You can also extract each channel of an input to specific outputs; the following command extracts two channels of the *INPUT* audio stream (file 0, stream 0) to the respective *OUTPUT_CH0* and *OUTPUT_CH1* outputs:

```
ffmpeg -i INPUT -map_channel 0.0.0 OUTPUT_CH0 -map_channel 0.0.1 OUTPUT_CH1
```

The following example splits the channels of a stereo input into two separate streams, which are put into the same output file:

```
ffmpeg -i stereo.wav -map 0:0 -map 0:0 -map_channel 0.0.0:0.0 -map_channel 0.0.1:0.1 -y out.ogg
```

Note that currently each output stream can only contain channels from a single input stream; you can't for example use "-map_channel" to pick multiple input audio channels contained in different streams (from the same or different files) and merge them into a single output stream. It is therefore not currently possible, for example, to turn two separate mono streams into a single stereo stream. However splitting a stereo stream into two single channel mono streams is possible.

If you need this feature, a possible workaround is to use the *amerge* filter. For example, if you need to merge a media (here ‘input.mkv’) with 2 mono audio streams into one single stereo channel audio stream (and keep the video stream), you can use the following command:

```
ffmpeg -i input.mkv -filter_complex "[0:1] [0:2] amerge" -c:a pcm_s16le -c:v copy output.mkv
```

‘**-map_metadata[:*metadata_spec_out*] *infile*[:*metadata_spec_in*] (*output*,*per-metadata*)**’

Set metadata information of the next output file from *infile*. Note that those are file indices (zero-based), not filenames. Optional *metadata_spec_in/out* parameters specify, which metadata to copy. A metadata specifier can have the following forms:

‘***g***’

global metadata, i.e. metadata that applies to the whole file

‘***s*[:*stream_spec*]**’

per-stream metadata. *stream_spec* is a stream specifier as described in the Stream specifiers chapter. In an input metadata specifier, the first matching stream is copied from. In an output metadata specifier, all matching streams are copied to.

‘***c*:*chapter_index***’

per-chapter metadata. *chapter_index* is the zero-based chapter index.

‘***p*:*program_index***’

per-program metadata. *program_index* is the zero-based program index.

If metadata specifier is omitted, it defaults to global.

By default, global metadata is copied from the first input file, per-stream and per-chapter metadata is copied along with streams/chapters. These default mappings are disabled by creating any mapping of the relevant type. A negative file index can be used to create a dummy mapping that just disables automatic copying.

For example to copy metadata from the first stream of the input file to global metadata of the output file:

```
ffmpeg -i in.ogg -map_metadata 0:s:0 out.mp3
```

To do the reverse, i.e. copy global metadata to all audio streams:

```
ffmpeg -i in.mkv -map_metadata:s:a 0:g out.mkv
```

Note that simple `0` would work as well in this example, since global metadata is assumed by default.

‘**-map_chapters *input_file_index* (*output*)**’

Copy chapters from input file with index *input_file_index* to the next output file. If no chapter mapping is specified, then chapters are copied from the first input file with at least one chapter. Use a negative file index to disable any chapter copying.

‘**-benchmark (*global*)**’

Show benchmarking information at the end of an encode. Shows CPU time used and maximum memory consumption. Maximum memory consumption is not supported on all systems, it will usually display as 0 if not supported.

‘**-benchmark_all (*global*)**’

Show benchmarking information during the encode. Shows CPU time used in various steps (audio/video encode/decode).

‘**-timelimit *duration* (*global*)**’

Exit after ffmpeg has been running for *duration* seconds.

‘**-dump (*global*)**’

Dump each input packet to stderr.

‘**-hex (*global*)**’

When dumping packets, also dump the payload.

‘**-re (*input*)**’

Read input at native frame rate. Mainly used to simulate a grab device. or live input stream (e.g. when reading from a

file). Should not be used with actual grab devices or live input streams (where it can cause packet loss). By default `ffmpeg` attempts to read the input(s) as fast as possible. This option will slow down the reading of the input(s) to the native frame rate of the input(s). It is useful for real-time output (e.g. live streaming).

'-loop_input'

Loop over the input stream. Currently it works only for image streams. This option is used for automatic FFserver testing. This option is deprecated, use -loop 1.

'-loop_output *number_of_times*'

Repeatedly loop output for formats that support looping such as animated GIF (0 will loop the output infinitely). This option is deprecated, use -loop.

'-vsync *parameter*'

Video sync method. For compatibility reasons old values can be specified as numbers. Newly added values will have to be specified as strings always.

'0, passthrough'

Each frame is passed with its timestamp from the demuxer to the muxer.

'1, cfr'

Frames will be duplicated and dropped to achieve exactly the requested constant frame rate.

'2, vfr'

Frames are passed through with their timestamp or dropped so as to prevent 2 frames from having the same timestamp.

'drop'

As passthrough but destroys all timestamps, making the muxer generate fresh timestamps based on frame-rate.

'-1, auto'

Chooses between 1 and 2 depending on muxer capabilities. This is the default method.

Note that the timestamps may be further modified by the muxer, after this. For example, in the case that the format option 'avoid_negative_ts' is enabled.

With -map you can select from which stream the timestamps should be taken. You can leave either video or audio unchanged and sync the remaining stream(s) to the unchanged one.

'-async *samples_per_second*'

Audio sync method. "Stretches/squeezes" the audio stream to match the timestamps, the parameter is the maximum samples per second by which the audio is changed. -async 1 is a special case where only the start of the audio stream is corrected without any later correction.

Note that the timestamps may be further modified by the muxer, after this. For example, in the case that the format option 'avoid_negative_ts' is enabled.

This option has been deprecated. Use the `aresample` audio filter instead.

'-copyts'

Do not process input timestamps, but keep their values without trying to sanitize them. In particular, do not remove the initial start time offset value.

Note that, depending on the 'vsync' option or on specific muxer processing (e.g. in case the format option 'avoid_negative_ts' is enabled) the output timestamps may mismatch with the input timestamps even when this option is selected.

'-copytb *mode*'

Specify how to set the encoder timebase when stream copying. *mode* is an integer numeric value, and can assume one of the following values:

'1'

Use the demuxer timebase.

The time base is copied to the output encoder from the corresponding input demuxer. This is sometimes required to avoid non monotonically increasing timestamps when copying video streams with variable frame rate.

'0'

Use the decoder timebase.

The time base is copied to the output encoder from the corresponding input decoder.

‘**-1**’

Try to make the choice automatically, in order to generate a sane output.

Default value is -1.

‘**-shortest (*output*)**’

Finish encoding when the shortest input stream ends.

‘**-dts_delta_threshold**’

Timestamp discontinuity delta threshold.

‘**-muxdelay *seconds* (*input*)**’

Set the maximum demux-decode delay.

‘**-muxpreload *seconds* (*input*)**’

Set the initial demux-decode delay.

‘**-streamid *output-stream-index*:*new-value* (*output*)**’

Assign a new stream-id value to an output stream. This option should be specified prior to the output filename to which it applies. For the situation where multiple output files exist, a streamid may be reassigned to a different value.

For example, to set the stream 0 PID to 33 and the stream 1 PID to 36 for an output mpegts file:

```
ffmpeg -i infile -streamid 0:33 -streamid 1:36 out.ts
```

‘**-bsf[:*stream_specifier*] *bitstream_filters* (*output,per-stream*)**’

Set bitstream filters for matching streams. *bitstream_filters* is a comma-separated list of bitstream filters. Use the `-bsfs` option to get the list of bitstream filters.

```
ffmpeg -i h264.mp4 -c:v copy -bsf:v h264_mp4toannexb -an out.h264
```

```
ffmpeg -i file.mov -an -vn -bsf:s mov2textsub -c:s copy -f rawvideo sub.txt
```

‘**-tag[:*stream_specifier*] *codec_tag* (*per-stream*)**’

Force a tag/fourcc for matching streams.

‘**-timecode *hh*:*mm*:*ss*SEP*ff***’

Specify Timecode for writing. *SEP* is ':' for non drop timecode and ';' (or '.') for drop.

```
ffmpeg -i input.mpg -timecode 01:02:03.04 -r 30000/1001 -s ntsc output.mpg
```

‘**-filter_complex *filtergraph* (*global*)**’

Define a complex filtergraph, i.e. one with arbitrary number of inputs and/or outputs. For simple graphs – those with one input and one output of the same type – see the ‘`-filter`’ options. *filtergraph* is a description of the filtergraph, as described in the “Filtergraph syntax” section of the ffmpeg-filters manual.

Input link labels must refer to input streams using the `[file_index:stream_specifier]` syntax (i.e. the same as ‘`-map`’ uses). If *stream_specifier* matches multiple streams, the first one will be used. An unlabeled input will be connected to the first unused input stream of the matching type.

Output link labels are referred to with ‘`-map`’. Unlabeled outputs are added to the first output file.

Note that with this option it is possible to use only lavfi sources without normal input files.

For example, to overlay an image over video

```
ffmpeg -i video.mkv -i image.png -filter_complex '[0:v][1:v]overlay[out]' -map
'[out]' out.mkv
```

Here `[0:v]` refers to the first video stream in the first input file, which is linked to the first (main) input of the overlay filter. Similarly the first video stream in the second input is linked to the second (overlay) input of overlay.

Assuming there is only one video stream in each input file, we can omit input labels, so the above is equivalent to

```
ffmpeg -i video.mkv -i image.png -filter_complex 'overlay[out]' -map
'[out]' out.mkv
```

Furthermore we can omit the output label and the single output from the filter graph will be added to the output file automatically, so we can simply write

```
ffmpeg -i video.mkv -i image.png -filter_complex 'overlay' out.mkv
```

To generate 5 seconds of pure red video using lavfi `color` source:

```
ffmpeg -filter_complex 'color=c=red' -t 5 out.mkv
```

'-lavfi *filtergraph* (*global*)'

Define a complex filtergraph, i.e. one with arbitrary number of inputs and/or outputs. Equivalent to '`-filter_complex`'.

'-filter_complex_script *filename* (*global*)'

This option is similar to '`-filter_complex`', the only difference is that its argument is the name of the file from which a complex filtergraph description is to be read.

'-accurate_seek (*input*)'

This option enables or disables accurate seeking in input files with the '`-ss`' option. It is enabled by default, so seeking is accurate when transcoding. Use '`-noaccurate_seek`' to disable it, which may be useful e.g. when copying some streams and transcoding the others.

'-override_ffserver (*global*)'

Overrides the input specifications from `ffserver`. Using this option you can map any input stream to `ffserver` and control many aspects of the encoding from `ffmpeg`. Without this option `ffmpeg` will transmit to `ffserver` what is requested by `ffserver`.

The option is intended for cases where features are needed that cannot be specified to `ffserver` but can be to `ffmpeg`.

As a special exception, you can use a bitmap subtitle stream as input: it will be converted into a video with the same size as the largest video in the file, or 720x576 if no video is present. Note that this is an experimental and temporary solution. It will be removed once libavfilter has proper support for subtitles.

For example, to hardcode subtitles on top of a DVB-T recording stored in MPEG-TS format, delaying the subtitles by 1 second:

```
ffmpeg -i input.ts -filter_complex \
  '[#0x2ef] setpts=PTS+1/TB [sub] ; [#0x2d0] [sub] overlay' \
  -sn -map '#0x2dc' output.mkv
```

(0x2d0, 0x2dc and 0x2ef are the MPEG-TS PIDs of respectively the video, audio and subtitles streams; 0:0, 0:3 and 0:7 would have worked too)

5.12 Preset files

A preset file contains a sequence of *option=value* pairs, one for each line, specifying a sequence of options which would be awkward to specify on the command line. Lines starting with the hash ('#') character are ignored and are used to provide comments. Check the '`presets`' directory in the FFmpeg source tree for examples.

Preset files are specified with the `vpre`, `apre`, `spre`, and `fpre` options. The `fpre` option takes the filename of the preset instead of a preset name as input and can be used for any kind of codec. For the `vpre`, `apre`, and `spre` options, the options specified in a preset file are applied to the currently selected codec of the same type as the preset option.

The argument passed to the `vpre`, `apre`, and `spre` preset options identifies the preset file to use according to the following rules:

First ffmpeg searches for a file named *arg*.ffpreset in the directories '`$FFMPEG_DATADIR`' (if set), and '`$HOME/.ffmpeg`', and in the datadir defined at configuration time (usually '`PREFIX/share/ffmpeg`') or in a '`ffpresets`' folder along the executable on win32, in that order. For example, if the argument is `libvpx-1080p`, it will search for the file '`libvpx-1080p.ffpreset`'.

If no such file is found, then ffmpeg will search for a file named *codec_name-arg*.ffpreset in the above-mentioned directories, where *codec_name* is the name of the codec to which the preset file options will be applied. For example, if you select the video codec with `-vcodec libvpx` and use `-vpre 1080p`, then it will search for the file '`libvpx-1080p.ffpreset`'.

6. Tips

- For streaming at very low bitrates, use a low frame rate and a small GOP size. This is especially true for RealVideo where the Linux player does not seem to be very fast, so it can miss frames. An example is:

```
ffmpeg -g 3 -r 3 -t 10 -b:v 50k -s qcif -f rv10 /tmp/b.rm
```

- The parameter 'q' which is displayed while encoding is the current quantizer. The value 1 indicates that a very good quality could be achieved. The value 31 indicates the worst quality. If q=31 appears too often, it means that the encoder cannot compress enough to meet your bitrate. You must either increase the bitrate, decrease the frame rate or decrease the frame size.
- If your computer is not fast enough, you can speed up the compression at the expense of the compression ratio. You can use '-me zero' to speed up motion estimation, and '-g 0' to disable motion estimation completely (you have only I-frames, which means it is about as good as JPEG compression).
- To have very low audio bitrates, reduce the sampling frequency (down to 22050 Hz for MPEG audio, 22050 or 11025 for AC-3).
- To have a constant quality (but a variable bitrate), use the option '-qscale n' when 'n' is between 1 (excellent quality) and 31 (worst quality).

7. Examples

7.1 Preset files

A preset file contains a sequence of *option=value* pairs, one for each line, specifying a sequence of options which can be specified also on the command line. Lines starting with the hash ('#') character are ignored and are used to provide comments. Empty lines are also ignored. Check the '`presets`' directory in the FFmpeg source tree for examples.

Preset files are specified with the `pre` option, this option takes a preset name as input. FFmpeg searches for a file named *preset_name*.avpreset in the directories '`$AVCONV_DATADIR`' (if set), and '`$HOME/.ffmpeg`', and in the data directory defined at configuration time (usually '`$PREFIX/share/ffmpeg`') in that order. For example, if the argument is `libx264-max`, it will search for the file '`libx264-max.avpreset`'.

7.2 Video and Audio grabbing

If you specify the input format and device then ffmpeg can grab video and audio directly.

```
ffmpeg -f oss -i /dev/dsp -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

Or with an ALSA audio source (mono input, card id 1) instead of OSS:

```
ffmpeg -f alsa -ac 1 -i hw:1 -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

Note that you must activate the right video source and channel before launching ffmpeg with any TV viewer such as xawtv by Gerd Knorr. You also have to set the audio recording levels correctly with a standard mixer.

7.3 X11 grabbing

Grab the X11 display with ffmpeg via

```
ffmpeg -f x11grab -video_size cif -framerate 25 -i :0.0 /tmp/out.mpg
```

0.0 is display.screen number of your X11 server, same as the DISPLAY environment variable.

```
ffmpeg -f x11grab -video_size cif -framerate 25 -i :0.0+10,20 /tmp/out.mpg
```

0.0 is display.screen number of your X11 server, same as the DISPLAY environment variable. 10 is the x-offset and 20 the y-offset for the grabbing.

7.4 Video and Audio file format conversion

Any supported file format and protocol can serve as input to ffmpeg:

Examples:

- You can use YUV files as input:

```
ffmpeg -i /tmp/test%d.Y /tmp/out.mpg
```

It will use the files:

```
/tmp/test0.Y, /tmp/test0.U, /tmp/test0.V,
/tmp/test1.Y, /tmp/test1.U, /tmp/test1.V, etc...
```

The Y files use twice the resolution of the U and V files. They are raw files, without header. They can be generated by all decent video decoders. You must specify the size of the image with the '`-s`' option if ffmpeg cannot guess it.

- You can input from a raw YUV420P file:

```
ffmpeg -i /tmp/test.yuv /tmp/out.avi
```

test.yuv is a file containing raw YUV planar data. Each frame is composed of the Y plane followed by the U and V planes at half vertical and horizontal resolution.

- You can output to a raw YUV420P file:

```
ffmpeg -i mydivx.avi hugefile.yuv
```

- You can set several input files and output files:

```
ffmpeg -i /tmp/a.wav -s 640x480 -i /tmp/a.yuv /tmp/a.mpg
```

Converts the audio file a.wav and the raw YUV video file a.yuv to MPEG file a.mpg.

- You can also do audio and video conversions at the same time:

```
ffmpeg -i /tmp/a.wav -ar 22050 /tmp/a.mp2
```

Converts a.wav to MPEG audio at 22050 Hz sample rate.

- You can encode to several formats at the same time and define a mapping from input stream to output streams:

```
ffmpeg -i /tmp/a.wav -map 0:a -b:a 64k /tmp/a.mp2 -map 0:a -b:a 128k /tmp/b.mp2
```

Converts a.wav to a.mp2 at 64 kbits and to b.mp2 at 128 kbits. '-map file:index' specifies which input stream is used for each output stream, in the order of the definition of output streams.

- You can transcode decrypted VOBs:

```
ffmpeg -i snatch_1.vob -f avi -c:v mpeg4 -b:v 800k -g 300 -bf 2 -c:a libmp3lame -b:a 128k snatch.avi
```

This is a typical DVD ripping example; the input is a VOB file, the output an AVI file with MPEG-4 video and MP3 audio. Note that in this command we use B-frames so the MPEG-4 stream is DivX5 compatible, and GOP size is 300 which means one intra frame every 10 seconds for 29.97fps input video. Furthermore, the audio stream is MP3-encoded so you need to enable LAME support by passing `--enable-libmp3lame` to configure. The mapping is particularly useful for DVD transcoding to get the desired audio language.

NOTE: To see the supported input formats, use `ffmpeg -formats`.

- You can extract images from a video, or create a video from many images:

For extracting images from a video:

```
ffmpeg -i foo.avi -r 1 -s WxH -f image2 foo-%03d.jpeg
```

This will extract one video frame per second from the video and will output them in files named '`foo-001.jpeg`', '`foo-002.jpeg`', etc. Images will be rescaled to fit the new WxH values.

If you want to extract just a limited number of frames, you can use the above command in combination with the -vframes or -t option, or in combination with -ss to start extracting from a certain point in time.

For creating a video from many images:

```
ffmpeg -f image2 -i foo-%03d.jpeg -r 12 -s WxH foo.avi
```

The syntax `foo-%03d.jpeg` specifies to use a decimal number composed of three digits padded with zeroes to express the sequence number. It is the same syntax supported by the C printf function, but only formats accepting a normal integer are suitable.

When importing an image sequence, -i also supports expanding shell-like wildcard patterns (globbing) internally, by selecting the image2-specific `-pattern_type glob` option.

For example, for creating a video from filenames matching the glob pattern `foo-*.jpeg`:

```
ffmpeg -f image2 -pattern_type glob -i 'foo-*.jpeg' -r 12 -s WxH foo.avi
```

- You can put many streams of the same type in the output:

```
ffmpeg -i test1.avi -i test2.avi -map 0:3 -map 0:2 -map 0:1 -map 0:0 -c copy test12.nut
```

  The resulting output file ‘test12.avi’ will contain first four streams from the input file in reverse order.

- To force CBR video output:

```
ffmpeg -i myfile.avi -b 4000k -minrate 4000k -maxrate 4000k -bufsize 1835k out.m2v
```

- The four options lmin, lmax, mblmin and mblmax use ’lambda’ units, but you may use the QP2LAMBDA constant to easily convert from ’q’ units:

```
ffmpeg -i src.ext -lmax 21*QP2LAMBDA dst.ext
```

8. See Also

ffmpeg-all, ffplay, ffprobe, ffserver, ffmpeg-utils, ffmpeg-scaler, ffmpeg-resampler, ffmpeg-codecs, ffmpeg-bitstream-filters, ffmpeg-formats, ffmpeg-devices, ffmpeg-protocols, ffmpeg-filters

9. Authors

The FFmpeg developers.

For details about the authorship, see the Git history of the project (git://source.ffmpeg.org/ffmpeg), e.g. by typing the command `git log` in the FFmpeg source directory, or browsing the online repository at http://source.ffmpeg.org.

Maintainers for the specific components are listed in the file ‘MAINTAINERS’ in the source code tree.

This document was generated on *January 20, 2014* using *texi2html 1.82*.

Get my process output in ErrorDataReceived instead of OutputDataReceived

1

in my application i am open Tshark process and start capturing and in order to update my UI i am want to get my process output but i am received this output in ErrorDataReceived instead of OutputDataReceived.

```
public void startCapturing()
{
    ProcessStartInfo tsharkStartInfo = new ProcessStartInfo();
    tsharkStartInfo.FileName = _tshark;
    tsharkStartInfo.RedirectStandardOutput = true;
    tsharkStartInfo.RedirectStandardError = true;
    tsharkStartInfo.RedirectStandardInput = true;
    tsharkStartInfo.UseShellExecute = false;
    tsharkStartInfo.CreateNoWindow = true;
    tsharkStartInfo.Arguments = string.Format(" -i " + _interfaceNumber +
                        " -s " + _packetLimitSize + " -w " + _pcapPath);
    _tsharkProcess.StartInfo = tsharkStartInfo;
    _tsharkProcess.ErrorDataReceived += _cmdProcess_ErrorDataReceived;
    _tsharkProcess.OutputDataReceived += tshark_OutputDataReceived;
    _tsharkProcess.EnableRaisingEvents = true;
    _tsharkProcess.Start();
    _tsharkProcess.BeginOutputReadLine();
    _tsharkProcess.BeginErrorReadLine();
    _tsharkProcess.WaitForExit();
}

void _cmdProcess_ErrorDataReceived(object sender, DataReceivedEventArgs e)
{
    int.TryParse(e.Data, out _numberOfPackets);
}

void tshark_OutputDataReceived(object sender, DataReceivedEventArgs e)
{
    string str = e.Data;
}
```

c# wireshark

share | edit | flag

edited Oct 12 '12 at 8:05
Kapil Khandelwal
9,114 10 19

asked Oct 12 '12 at 8:02
Dana Yeger
61 1 8

add comment

tagged
c# × 578207
wireshark × 789

asked **1 year ago**
viewed **368 times**

Related

1 Why do I no longer receive OutputDataReceived events when restarting a process?

3 Capturing process output via OutputDataReceived event

4 C# get process output while running

1 How to get the output of a process?

0 Add wireshark into my code instead of install on the machine

0 Application stuck when get process output because of high speed network

0 Get all process output even after process killed

0 Count all received packet using Tshark

0 trying to open Tshark process and start capturing

0 Tshark stop capturing after few minutes although process still running

Hot Network Questions

Why there is such a big difference between "Size" and "Size on disk"?

Post Your Answer

Browse other questions tagged c# wireshark or ask your own question.

- How do I explain to a 6 year old why people on the other side of the earth don't fall off?
- The Spiral of Roots in TikZ
- What is the difference between a point and a vector
- Is it ethical to write answers to work-relevant Stack Exchange questions on the clock?

more hot questions

question feed

about help blog chat data legal privacy policy jobs advertising info mobile **contact us** **feedback**

TECHNOLOGY

Stack Overflow
Server Fault
Super User
Web Applications
Ask Ubuntu
Webmasters
Game Development
TeX - LaTeX
Programmers
Unix & Linux
Ask Different (Apple)
WordPress Answers
Geographic Information Systems
Electrical Engineering
Android Enthusiasts
Information Security
Database Administrators
Drupal Answers
SharePoint
User Experience
Mathematica
more (14)

LIFE / ARTS

Photography
Science Fiction & Fantasy
Seasoned Advice (cooking)
Home Improvement
more (13)

CULTURE / RECREATION

English Language & Usage
Skeptics
M Yodeya (Judaism)
Travel
Christianity
Arqade (gaming)
Bicycles
Role-playing Games
more (21)

SCIENCE

Mathematics
Cross Validated (stats)
Theoretical Computer Science
Physics
MathOverflow
more (7)

OTHER

Stack Apps
Meta Stack Ove
Area 51
Stack Overflow Careers

ffprobe Documentation

Table of Contents

1. Synopsis

ffprobe [*options*] [‘`input_file`’]

2. Description

ffprobe gathers information from multimedia streams and prints it in human- and machine-readable fashion.

For example it can be used to check the format of the container used by a multimedia stream and the format and type of each media stream contained in it.

If a filename is specified in input, ffprobe will try to open and probe the file content. If the file cannot be opened or recognized as a multimedia file, a positive exit code is returned.

ffprobe may be employed both as a standalone application or in combination with a textual filter, which may perform more sophisticated processing, e.g. statistical processing or plotting.

Options are used to list some of the formats supported by ffprobe or for specifying which information to display, and for setting how ffprobe will show it.

ffprobe output is designed to be easily parsable by a textual filter, and consists of one or more sections of a form defined by the selected writer, which is specified by the ‘`print_format`’ option.

Sections may contain other nested sections, and are identified by a name (which may be shared by other sections), and an unique name. See the output of ‘`sections`’.

Metadata tags stored in the container or in the streams are recognized and printed in the corresponding "FORMAT", "STREAM" or "PROGRAM_STREAM" section.

3. Options

All the numerical options, if not specified otherwise, accept a string representing a number as input, which may be followed by one of the SI unit prefixes, for example: ’K’, ’M’, or ’G’.

If ’i’ is appended to the SI unit prefix, the complete prefix will be interpreted as a unit prefix for binary multiplies, which are based on powers of 1024 instead of powers of 1000. Appending ’B’ to the SI unit prefix multiplies the value by 8. This allows using, for example: ’KB’, ’MiB’, ’G’ and ’B’ as number suffixes.

Options which do not take arguments are boolean options, and set the corresponding value to true. They can be set to false by prefixing the option name with "no". For example using "-nofoo" will set the boolean option with name "foo" to false.

3.1 Stream specifiers

Some options are applied per-stream, e.g. bitrate or codec. Stream specifiers are used to precisely specify which stream(s) a given option belongs to.

A stream specifier is a string generally appended to the option name and separated from it by a colon. E.g. `-codec:a:1 ac3` contains the `a:1` stream specifier, which matches the second audio stream. Therefore, it would select the ac3 codec for the second audio stream.

A stream specifier can match several streams, so that the option is applied to all of them. E.g. the stream specifier in `-b:a 128k` matches all audio streams.

An empty stream specifier matches all streams. For example, `-codec copy` or `-codec: copy` would copy all the streams without reencoding.

Possible forms of stream specifiers are:

‘*stream_index*’

Matches the stream with this index. E.g. `-threads:1 4` would set the thread count for the second stream to 4.

‘*stream_type*[:*stream_index*]’

stream_type is one of following: ’v’ for video, ’a’ for audio, ’s’ for subtitle, ’d’ for data, and ’t’ for attachments. If *stream_index* is given, then it matches stream number *stream_index* of this type. Otherwise, it matches all streams of this type.

‘p:*program_id*[:*stream_index*]’

If *stream_index* is given, then it matches the stream with number *stream_index* in the program with the id *program_id*. Otherwise, it matches all streams in the program.

‘#*stream_id*’

Matches the stream by a format-specific ID.

3.2 Generic options

These options are shared amongst the ff* tools.

‘-L’

Show license.

‘-h, -?, -help, --help [*arg*]’

Show help. An optional parameter may be specified to print help about a specific item. If no argument is specified, only basic (non advanced) tool options are shown.

Possible values of *arg* are:

‘long’

Print advanced tool options in addition to the basic tool options.

‘full’

Print complete list of options, including shared and private options for encoders, decoders, demuxers, muxers, filters, etc.

‘decoder=*decoder_name*’

Print detailed information about the decoder named *decoder_name*. Use the ‘-decoders’ option to get a list of all decoders.

‘encoder=*encoder_name*’

Print detailed information about the encoder named *encoder_name*. Use the ‘-encoders’ option to get a list of all encoders.

‘demuxer=*demuxer_name*’

Print detailed information about the demuxer named *demuxer_name*. Use the ‘-formats’ option to get a list of all demuxers and muxers.

‘muxer=*muxer_name*’

Print detailed information about the muxer named *muxer_name*. Use the ‘-formats’ option to get a list of all muxers and

demuxers.

‘filter=*filter_name*’

Print detailed information about the filter name *filter_name*. Use the ‘-filters’ option to get a list of all filters.

‘-version’

Show version.

‘-formats’

Show available formats.

‘-codecs’

Show all codecs known to libavcodec.

Note that the term ’codec’ is used throughout this documentation as a shortcut for what is more correctly called a media bitstream format.

‘-decoders’

Show available decoders.

‘-encoders’

Show all available encoders.

‘-bsfs’

Show available bitstream filters.

‘-protocols’

Show available protocols.

‘-filters’

Show available libavfilter filters.

‘-pix_fmts’

Show available pixel formats.

‘-sample_fmts’

Show available sample formats.

‘-layouts’

Show channel names and standard channel layouts.

‘-colors’

Show recognized color names.

‘-loglevel [repeat+]*loglevel* | -v [repeat+]*loglevel*’

Set the logging level used by the library. Adding "repeat+" indicates that repeated log output should not be compressed to the first line and the "Last message repeated n times" line will be omitted. "repeat" can also be used alone. If "repeat" is used alone, and with no prior loglevel set, the default loglevel will be used. If multiple loglevel parameters are given, using ’repeat’ will not change the loglevel. *loglevel* is a number or a string containing one of the following values:

‘quiet’

Show nothing at all; be silent.

‘panic’

Only show fatal errors which could lead the process to crash, such as and assert failure. This is not currently used for anything.

‘fatal’

Only show fatal errors. These are errors after which the process absolutely cannot continue after.

‘error’

Show all errors, including ones which can be recovered from.

‘warning’

Show all warnings and errors. Any message related to possibly incorrect or unexpected events will be shown.

‘info’

Show informative messages during processing. This is in addition to warnings and errors. This is the default value.

‘verbose’

Same as info, except more verbose.

‘debug’

Show everything, including debugging information.

By default the program logs to stderr, if coloring is supported by the terminal, colors are used to mark errors and warnings. Log coloring can be disabled setting the environment variable AV_LOG_FORCE_NOCOLOR or NO_COLOR, or can be forced setting the environment variable AV_LOG_FORCE_COLOR. The use of the environment variable NO_COLOR is deprecated and will be dropped in a following FFmpeg version.

‘-report’

Dump full command line and console output to a file named *program*-*YYYYMMDD*-*HHMMSS*.log in the current directory. This file can be useful for bug reports. It also implies -loglevel verbose.

Setting the environment variable FFREPORT to any value has the same effect. If the value is a ':'-separated key=value sequence, these options will affect the report; options values must be escaped if they contain special characters or the options delimiter ':' (see the “Quoting and escaping” section in the ffmpeg-utils manual). The following option is recognized:

‘file’

set the file name to use for the report; %p is expanded to the name of the program, %t is expanded to a timestamp, %% is expanded to a plain %

Errors in parsing the environment variable are not fatal, and will not appear in the report.

‘-cpuflags flags (*global*)’

Allows setting and clearing cpu flags. This option is intended for testing. Do not use it unless you know what you’re doing.

```
ffmpeg -cpuflags -sse+mmx ...
ffmpeg -cpuflags mmx ...
ffmpeg -cpuflags 0 ...
```

Possible flags for this option are:

‘x86’

‘mmx’
‘mmxext’
‘sse’
‘sse2’
‘sse2slow’
‘sse3’
‘sse3slow’
‘ssse3’
‘atom’
‘sse4.1’
‘sse4.2’
‘avx’
‘xop’
‘fma4’
‘3dnow’
‘3dnowext’
‘cmov’

‘ARM’

‘armv5te’
‘armv6’
‘armv6t2’
‘vfp’
‘vfpv3’
‘neon’

‘PowerPC’

‘altivec’

‘Specific Processors’

‘pentium2’
‘pentium3’
‘pentium4’
‘k6’
‘k62’
‘athlon’
‘athlonxp’
‘k8’

‘-opencl_options options (*global*)’

Set OpenCL environment options. This option is only available when FFmpeg has been compiled with `--enable-opencl`.

options must be a list of *key=value* option pairs separated by ’:’. See the “OpenCL Options” section in the ffmpeg-utils manual for the list of supported options.

3.3 AVOptions

These options are provided directly by the libavformat, libavdevice and libavcodec libraries. To see the list of available AVOptions, use the ‘`-help`’ option. They are separated into two categories:

‘generic’

These options can be set for any container, codec or device. Generic options are listed under AVFormatContext options for containers/devices and under AVCodecContext options for codecs.

‘private’

These options are specific to the given container, device or codec. Private options are listed under their corresponding containers/devices/codecs.

For example to write an ID3v2.3 header instead of a default ID3v2.4 to an MP3 file, use the ‘`id3v2_version`’ private option of the MP3 muxer:

```
ffmpeg -i input.flac -id3v2_version 3 out.mp3
```

All codec AVOptions are per-stream, and thus a stream specifier should be attached to them.

Note: the ‘`-nooption`’ syntax cannot be used for boolean AVOptions, use ‘`-option 0`’/‘`-option 1`’.

Note: the old undocumented way of specifying per-stream AVOptions by prepending v/a/s to the options name is now obsolete and will be removed soon.

3.4 Main options

‘`-f format`’

Force format to use.

‘`-unit`’

Show the unit of the displayed values.

‘`-prefix`’

Use SI prefixes for the displayed values. Unless the "-byte_binary_prefix" option is used all the prefixes are decimal.

‘-byte_binary_prefix’

Force the use of binary prefixes for byte values.

‘-sexagesimal’

Use sexagesimal format HH:MM:SS.MICROSECONDS for time values.

‘-pretty’

Prettify the format of the displayed values, it corresponds to the options "-unit -prefix -byte_binary_prefix -sexagesimal".

‘-of, -print_format *writer_name*[=*writer_options*]’

Set the output printing format.

writer_name specifies the name of the writer, and *writer_options* specifies the options to be passed to the writer.

For example for printing the output in JSON format, specify:

```
-print_format json
```

For more details on the available output printing formats, see the Writers section below.

‘-sections’

Print sections structure and section information, and exit. The output is not meant to be parsed by a machine.

‘-select_streams *stream_specifier*’

Select only the streams specified by *stream_specifier*. This option affects only the options related to streams (e.g. show_streams, show_packets, etc.).

For example to show only audio streams, you can use the command:

```
ffprobe -show_streams -select_streams a INPUT
```

To show only video packets belonging to the video stream with index 1:

```
ffprobe -show_packets -select_streams v:1 INPUT
```

‘-show_data’

Show payload data, as a hexadecimal and ASCII dump. Coupled with ‘-show_packets’, it will dump the packets’ data. Coupled with ‘-show_streams’, it will dump the codec extradata.

The dump is printed as the "data" field. It may contain newlines.

‘-show_error’

Show information about the error found when trying to probe the input.

The error information is printed within a section with name "ERROR".

‘-show_format’

Show information about the container format of the input multimedia stream.

All the container format information is printed within a section with name "FORMAT".

‘-show_format_entry *name*’

Like ‘-show_format’, but only prints the specified entry of the container format information, rather than all. This option may be given more than once, then all specified entries will be shown.

This option is deprecated, use show_entries instead.

‘-show_entries *section_entries*’

Set list of entries to show.

Entries are specified according to the following syntax. *section_entries* contains a list of section entries separated by `:`. Each section entry is composed by a section name (or unique name), optionally followed by a list of entries local to that section, separated by `,`.

If section name is specified but is followed by no `=`, all entries are printed to output, together with all the contained sections. Otherwise only the entries specified in the local section entries list are printed. In particular, if `=` is specified but the list of local entries is empty, then no entries will be shown for that section.

Note that the order of specification of the local section entries is not honored in the output, and the usual display order will be retained.

The formal syntax is given by:

```
LOCAL_SECTION_ENTRIES ::= SECTION_ENTRY_NAME[,LOCAL_SECTION_ENTRIES]
SECTION_ENTRY         ::= SECTION_NAME[=[LOCAL_SECTION_ENTRIES]]
SECTION_ENTRIES       ::= SECTION_ENTRY[:SECTION_ENTRIES]
```

For example, to show only the index and type of each stream, and the PTS time, duration time, and stream index of the packets, you can specify the argument:

```
packet=pts_time,duration_time,stream_index : stream=index,codec_type
```

To show all the entries in the section "format", but only the codec type in the section "stream", specify the argument:

```
format : stream=codec_type
```

To show all the tags in the stream and format sections:

```
format_tags : format_tags
```

To show only the `title` tag (if available) in the stream sections:

```
stream_tags=title
```

‘`-show_packets`’

Show information about each packet contained in the input multimedia stream.

The information for each single packet is printed within a dedicated section with name "PACKET".

‘`-show_frames`’

Show information about each frame and subtitle contained in the input multimedia stream.

The information for each single frame is printed within a dedicated section with name "FRAME" or "SUBTITLE".

‘`-show_streams`’

Show information about each media stream contained in the input multimedia stream.

Each media stream information is printed within a dedicated section with name "STREAM".

‘`-show_programs`’

Show information about programs and their streams contained in the input multimedia stream.

Each media stream information is printed within a dedicated section with name "PROGRAM_STREAM".

‘`-show_chapters`’

Show information about chapters stored in the format.

Each chapter is printed within a dedicated section with name "CHAPTER".

‘`-count_frames`’

Count the number of frames per stream and report it in the corresponding stream section.

‘`-count_packets`’

Count the number of packets per stream and report it in the corresponding stream section.

‘-read_intervals *read_intervals*’

Read only the specified intervals. *read_intervals* must be a sequence of interval specifications separated by ",". ffprobe will seek to the interval starting point, and will continue reading from that.

Each interval is specified by two optional parts, separated by "%".

The first part specifies the interval start position. It is interpreted as an abolute position, or as a relative offset from the current position if it is preceded by the "+" character. If this first part is not specified, no seeking will be performed when reading this interval.

The second part specifies the interval end position. It is interpreted as an absolute position, or as a relative offset from the current position if it is preceded by the "+" character. If the offset specification starts with "#", it is interpreted as the number of packets to read (not including the flushing packets) from the interval start. If no second part is specified, the program will read until the end of the input.

Note that seeking is not accurate, thus the actual interval start point may be different from the specified position. Also, when an interval duration is specified, the absolute end time will be computed by adding the duration to the interval start point found by seeking the file, rather than to the specified start value.

The formal syntax is given by:

```
INTERVAL  ::= [START|+START_OFFSET][%[END|+END_OFFSET]]
INTERVALS ::= INTERVAL[,INTERVALS]
```

A few examples follow.

- Seek to time 10, read packets until 20 seconds after the found seek point, then seek to position 01:30 (1 minute and thirty seconds) and read packets until position 01:45.

  ```
  10%+20,01:30%01:45
  ```

- Read only 42 packets after seeking to position 01:23:

  ```
  01:23%+#42
  ```

- Read only the first 20 seconds from the start:

  ```
  %+20
  ```

- Read from the start until position 02:30:

  ```
  %02:30
  ```

‘-show_private_data, -private’

Show private data, that is data depending on the format of the particular shown element. This option is enabled by default, but you may need to disable it for specific uses, for example when creating XSD-compliant XML output.

‘-show_program_version’

Show information related to program version.

Version information is printed within a section with name "PROGRAM_VERSION".

‘-show_library_versions’

Show information related to library versions.

Version information for each library is printed within a section with name "LIBRARY_VERSION".

‘-show_versions’

Show information related to program and library versions. This is the equivalent of setting both ‘-show_program_version’ and ‘-show_library_versions’ options.

‘-bitexact’

Force bitexact output, useful to produce output which is not dependent on the specific build.

‘-i *input_file*’

Read *input_file*.

4. Writers

A writer defines the output format adopted by `ffprobe`, and will be used for printing all the parts of the output.

A writer may accept one or more arguments, which specify the options to adopt. The options are specified as a list of *key=value* pairs, separated by ":".

All writers support the following options:

‘`string_validation, sv`’

Set string validation mode.

The following values are accepted.

‘`fail`’

The writer will fail immediately in case an invalid string (UTF-8) sequence or code point is found in the input. This is especially useful to validate input metadata.

‘`ignore`’

Any validation error will be ignored. This will result in possibly broken output, especially with the json or xml writer.

‘`replace`’

The writer will substitute invalid UTF-8 sequences or code points with the string specified with the ‘`string_validation_replacement`’.

Default value is ‘`replace`’.

‘`string_validation_replacement, svr`’

Set replacement string to use in case ‘`string_validation`’ is set to ‘`replace`’.

In case the option is not specified, the writer will assume the empty string, that is it will remove the invalid sequences from the input strings.

A description of the currently available writers follows.

4.1 default

Default format.

Print each section in the form:

```
[SECTION]
key1=val1
...
keyN=valN
[/SECTION]
```

Metadata tags are printed as a line in the corresponding FORMAT, STREAM or PROGRAM_STREAM section, and are prefixed by the string "TAG:".

A description of the accepted options follows.

‘`nokey, nk`’

If set to 1 specify not to print the key of each field. Default value is 0.

‘`noprint_wrappers, nw`’

If set to 1 specify not to print the section header and footer. Default value is 0.

4.2 compact, csv

Compact and CSV format.

The `csv` writer is equivalent to `compact`, but supports different defaults.

Each section is printed on a single line. If no option is specifid, the output has the form:

```
section|key1=val1| ... |keyN=valN
```

Metadata tags are printed in the corresponding "format" or "stream" section. A metadata tag key, if printed, is prefixed by the string "tag:".

The description of the accepted options follows.

‘`item_sep, s`’

Specify the character to use for separating fields in the output line. It must be a single printable character, it is "|" by default ("," for the `csv` writer).

‘`nokey, nk`’

If set to 1 specify not to print the key of each field. Its default value is 0 (1 for the `csv` writer).

‘`escape, e`’

Set the escape mode to use, default to "c" ("csv" for the `csv` writer).

It can assume one of the following values:

‘`c`’

Perform C-like escaping. Strings containing a newline (’\n’), carriage return (’\r’), a tab (’\t’), a form feed (’\f’), the escaping character (’\’) or the item separator character *SEP* are escaped using C-like fashioned escaping, so that a newline is converted to the sequence "\n", a carriage return to "\r", ’\’ to "\\" and the separator *SEP* is converted to "*SEP*".

‘`csv`’

Perform CSV-like escaping, as described in RFC4180. Strings containing a newline (’\n’), a carriage return (’\r’), a double quote (’"’), or *SEP* are enclosed in double-quotes.

‘`none`’

Perform no escaping.

‘`print_section, p`’

Print the section name at the begin of each line if the value is `1`, disable it with value set to `0`. Default value is `1`.

4.3 flat

Flat format.

A free-form output where each line contains an explicit key=value, such as "streams.stream.3.tags.foo=bar". The output is shell escaped, so it can be directly embedded in sh scripts as long as the separator character is an alphanumeric character or an underscore (see *sep_char* option).

The description of the accepted options follows.

‘`sep_char, s`’

Separator character used to separate the chapter, the section name, IDs and potential tags in the printed field key.

Default value is ’.’.

‘`hierarchical, h`’

Specify if the section name specification should be hierarchical. If set to 1, and if there is more than one section in the current chapter, the section name will be prefixed by the name of the chapter. A value of 0 will disable this behavior.

Default value is 1.

4.4 ini

INI format output.

Print output in an INI based format.

The following conventions are adopted:

- all key and values are UTF-8
- '.' is the subgroup separator
- newline, '\t', '\f', '\b' and the following characters are escaped
- '\' is the escape character
- '#' is the comment indicator
- '=' is the key/value separator
- ':' is not used but usually parsed as key/value separator

This writer accepts options as a list of *key=value* pairs, separated by ":".

The description of the accepted options follows.

'hierarchical, h'

Specify if the section name specification should be hierarchical. If set to 1, and if there is more than one section in the current chapter, the section name will be prefixed by the name of the chapter. A value of 0 will disable this behavior.

Default value is 1.

4.5 json

JSON based format.

Each section is printed using JSON notation.

The description of the accepted options follows.

'compact, c'

If set to 1 enable compact output, that is each section will be printed on a single line. Default value is 0.

For more information about JSON, see http://www.json.org/.

4.6 xml

XML based format.

The XML output is described in the XML schema description file 'ffprobe.xsd' installed in the FFmpeg datadir.

An updated version of the schema can be retrieved at the url http://www.ffmpeg.org/schema/ffprobe.xsd, which redirects to the latest schema committed into the FFmpeg development source code tree.

Note that the output issued will be compliant to the 'ffprobe.xsd' schema only when no special global output options ('unit', 'prefix', 'byte_binary_prefix', 'sexagesimal' etc.) are specified.

The description of the accepted options follows.

'fully_qualified, q'

If set to 1 specify if the output should be fully qualified. Default value is 0. This is required for generating an XML file which can be validated through an XSD file.

'xsd_compliant, x'

If set to 1 perform more checks for ensuring that the output is XSD compliant. Default value is 0. This option automatically sets 'fully_qualified' to 1.

For more information about the XML format, see http://www.w3.org/XML/.

5. Timecode

ffprobe supports Timecode extraction:

- MPEG1/2 timecode is extracted from the GOP, and is available in the video stream details ('-show_streams', see *timecode*).
- MOV timecode is extracted from tmcd track, so is available in the tmcd stream metadata ('-show_streams', see *TAG:timecode*).
- DV, GXF and AVI timecodes are available in format metadata ('-show_format', see *TAG:timecode*).

6. See Also

ffprobe-all, ffmpeg, ffplay, ffserver, ffmpeg-utils, ffmpeg-scaler, ffmpeg-resampler, ffmpeg-codecs, ffmpeg-bitstream-filters, ffmpeg-formats, ffmpeg-devices, ffmpeg-protocols, ffmpeg-filters

7. Authors

The FFmpeg developers.

For details about the authorship, see the Git history of the project (git://source.ffmpeg.org/ffmpeg), e.g. by typing the command `git log` in the FFmpeg source directory, or browsing the online repository at http://source.ffmpeg.org.

Maintainers for the specific components are listed in the file '`MAINTAINERS`' in the source code tree.

This document was generated by *Kyle Schwarz* on *December 1, 2013* using *texi2html 1.82*.

ffmpeg Documentation

Table of Contents

1. Synopsis

ffmpeg [*global_options*] {[*input_file_options*] -i '`input_file`'} ... {[*output_file_options*] '`output_file`'} ...

2. Description

`ffmpeg` is a very fast video and audio converter that can also grab from a live audio/video source. It can also convert between arbitrary sample rates and resize video on the fly with a high quality polyphase filter.

`ffmpeg` reads from an arbitrary number of input "files" (which can be regular files, pipes, network streams, grabbing devices, etc.), specified by the `-i` option, and writes to an arbitrary number of output "files", which are specified by a plain output filename. Anything found on the command line which cannot be interpreted as an option is considered to be an output filename.

Each input or output file can, in principle, contain any number of streams of different types (video/audio/subtitle/attachment/data). The allowed number and/or types of streams may be limited by the container format. Selecting which streams from which inputs will go into which output is either done automatically or with the `-map` option (see the Stream selection chapter).

To refer to input files in options, you must use their indices (0-based). E.g. the first input file is `0`, the second is `1`, etc. Similarly, streams within a file are referred to by their indices. E.g. `2:3` refers to the fourth stream in the third input file. Also see the Stream specifiers chapter.

As a general rule, options are applied to the next specified file. Therefore, order is important, and you can have the same option on the command line multiple times. Each occurrence is then applied to the next input or output file. Exceptions from this rule are the global options (e.g. verbosity level), which should be specified first.

Do not mix input and output files – first specify all input files, then all output files. Also do not mix options which belong to different files. All options apply ONLY to the next input or output file and are reset between files.

- To set the video bitrate of the output file to 64 kbit/s:

```
ffmpeg -i input.avi -b:v 64k -bufsize 64k output.avi
```

- To force the frame rate of the output file to 24 fps:

```
ffmpeg -i input.avi -r 24 output.avi
```

- To force the frame rate of the input file (valid for raw formats only) to 1 fps and the frame rate of the output file to 24 fps:

```
ffmpeg -r 1 -i input.m2v -r 24 output.avi
```

The format option may be needed for raw input files.

3. Detailed description

The transcoding process in `ffmpeg` for each output can be described by the following diagram:

`ffmpeg` calls the libavformat library (containing demuxers) to read input files and get packets containing encoded data from them. When there are multiple input files, `ffmpeg` tries to keep them synchronized by tracking lowest timestamp on any active input stream.

Encoded packets are then passed to the decoder (unless streamcopy is selected for the stream, see further for a description). The decoder produces uncompressed frames (raw video/PCM audio/...) which can be processed further by filtering (see next section). After filtering, the frames are passed to the encoder, which encodes them and outputs encoded packets. Finally those are passed to the muxer, which writes the encoded packets to the output file.

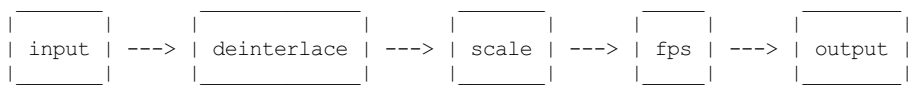
3.1 Filtering

Before encoding, `ffmpeg` can process raw audio and video frames using filters from the libavfilter library. Several chained filters form a filter graph. `ffmpeg` distinguishes between two types of filtergraphs: simple and complex.

3.1.1 Simple filtergraphs

Simple filtergraphs are those that have exactly one input and output, both of the same type. In the above diagram they can be represented by simply inserting an additional step between decoding and encoding:

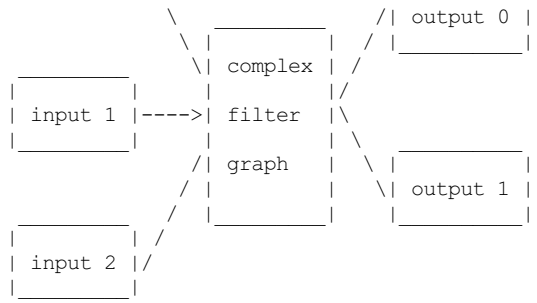
Simple filtergraphs are configured with the per-stream '`-filter`' option (with '`-vf`' and '`-af`' aliases for video and audio respectively). A simple filtergraph for video can look for example like this:



Note that some filters change frame properties but not frame contents. E.g. the `fps` filter in the example above changes number of frames, but does not touch the frame contents. Another example is the `setpts` filter, which only sets timestamps and otherwise passes the frames unchanged.

3.1.2 Complex filtergraphs

Complex filtergraphs are those which cannot be described as simply a linear processing chain applied to one stream. This is the case, for example, when the graph has more than one input and/or output, or when output stream type is different from input. They can be represented with the following diagram:



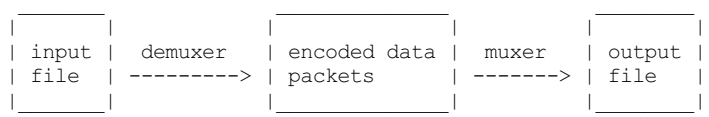
Complex filtergraphs are configured with the '`-filter_complex`' option. Note that this option is global, since a complex filtergraph, by its nature, cannot be unambiguously associated with a single stream or file.

The '`-lavfi`' option is equivalent to '`-filter_complex`'.

A trivial example of a complex filtergraph is the `overlay` filter, which has two video inputs and one video output, containing one video overlaid on top of the other. Its audio counterpart is the `amix` filter.

3.2 Stream copy

Stream copy is a mode selected by supplying the `copy` parameter to the '`-codec`' option. It makes `ffmpeg` omit the decoding and encoding step for the specified stream, so it does only demuxing and muxing. It is useful for changing the container format or modifying container-level metadata. The diagram above will, in this case, simplify to this:



Since there is no decoding or encoding, it is very fast and there is no quality loss. However, it might not work in some cases because of many factors. Applying filters is obviously also impossible, since filters work on uncompressed data.

4. Stream selection

By default, `ffmpeg` includes only one stream of each type (video, audio, subtitle) present in the input files and adds them to each output file. It picks the "best" of each based upon the following criteria: for video, it is the stream with the highest resolution, for audio, it is the stream with the most channels, for subtitles, it is the first subtitle stream. In the case where several streams of the same type rate equally, the stream with the lowest index is chosen.

You can disable some of those defaults by using the `-vn/-an/-sn` options. For full manual control, use the `-map` option, which disables the defaults just described.

5. Options

All the numerical options, if not specified otherwise, accept a string representing a number as input, which may be followed by one of the SI unit prefixes, for example: 'K', 'M', or 'G'.

If 'i' is appended to the SI unit prefix, the complete prefix will be interpreted as a unit prefix for binary multiplies, which are based on powers of 1024 instead of powers of 1000. Appending 'B' to the SI unit prefix multiplies the value by 8. This allows using, for example: 'KB', 'MiB', 'G' and 'B' as number suffixes.

Options which do not take arguments are boolean options, and set the corresponding value to true. They can be set to false by prefixing the option name with "no". For example using "-nofoo" will set the boolean option with name "foo" to false.

5.1 Stream specifiers

Some options are applied per-stream, e.g. bitrate or codec. Stream specifiers are used to precisely specify which stream(s) a given option belongs to.

A stream specifier is a string generally appended to the option name and separated from it by a colon. E.g. `-codec:a:1 ac3` contains the `a:1` stream specifier, which matches the second audio stream. Therefore, it would select the ac3 codec for the second audio stream.

A stream specifier can match several streams, so that the option is applied to all of them. E.g. the stream specifier in `-b:a 128k` matches all audio streams.

An empty stream specifier matches all streams. For example, `-codec copy` or `-codec: copy` would copy all the streams without reencoding.

Possible forms of stream specifiers are:

‘*stream_index*’

Matches the stream with this index. E.g. `-threads:1 4` would set the thread count for the second stream to 4.

‘*stream_type*[:*stream_index*]’

stream_type is one of following: ’v’ for video, ’a’ for audio, ’s’ for subtitle, ’d’ for data, and ’t’ for attachments. If *stream_index* is given, then it matches stream number *stream_index* of this type. Otherwise, it matches all streams of this type.

‘p:*program_id*[:*stream_index*]’

If *stream_index* is given, then it matches the stream with number *stream_index* in the program with the id *program_id*. Otherwise, it matches all streams in the program.

‘#*stream_id*’

Matches the stream by a format-specific ID.

5.2 Generic options

These options are shared amongst the ff* tools.

‘-L’

Show license.

‘-h, -?, -help, --help [*arg*]’

Show help. An optional parameter may be specified to print help about a specific item. If no argument is specified, only basic (non advanced) tool options are shown.

Possible values of *arg* are:

‘long’

Print advanced tool options in addition to the basic tool options.

‘full’

Print complete list of options, including shared and private options for encoders, decoders, demuxers, muxers, filters, etc.

‘decoder=*decoder_name*’

Print detailed information about the decoder named *decoder_name*. Use the ‘-decoders’ option to get a list of all decoders.

‘encoder=*encoder_name*’

Print detailed information about the encoder named *encoder_name*. Use the ‘-encoders’ option to get a list of all encoders.

‘demuxer=*demuxer_name*’

Print detailed information about the demuxer named *demuxer_name*. Use the ‘-formats’ option to get a list of all demuxers and muxers.

‘muxer=*muxer_name*’

Print detailed information about the muxer named *muxer_name*. Use the ‘-formats’ option to get a list of all muxers and demuxers.

‘filter=*filter_name*’

Print detailed information about the filter name *filter_name*. Use the ‘-filters’ option to get a list of all filters.

‘-version’

Show version.

‘-formats’

Show available formats.

‘-codecs’

Show all codecs known to libavcodec.

Note that the term ’codec’ is used throughout this documentation as a shortcut for what is more correctly called a media bitstream format.

‘-decoders’

Show available decoders.

‘-encoders’

Show all available encoders.

‘-bsfs’

Show available bitstream filters.

‘-protocols’

Show available protocols.

‘-filters’

Show available libavfilter filters.

‘-pix_fmts’

Show available pixel formats.

‘-sample_fmts’

Show available sample formats.

‘-layouts’

Show channel names and standard channel layouts.

‘-colors’

Show recognized color names.

‘-loglevel [repeat+]*loglevel* | -v [repeat+]*loglevel*’

Set the logging level used by the library. Adding "repeat+" indicates that repeated log output should not be compressed to the first line and the "Last message repeated n times" line will be omitted. "repeat" can also be used alone. If "repeat" is used alone, and with no prior loglevel set, the default loglevel will be used. If multiple loglevel parameters are given, using ’repeat’ will not change the loglevel. *loglevel* is a number or a string containing one of the following values:

‘quiet’

Show nothing at all; be silent.

‘panic’

Only show fatal errors which could lead the process to crash, such as and assert failure. This is not currently used for anything.

‘fatal’

Only show fatal errors. These are errors after which the process absolutely cannot continue after.

‘error’

Show all errors, including ones which can be recovered from.

‘warning’

Show all warnings and errors. Any message related to possibly incorrect or unexpected events will be shown.

‘info’

Show informative messages during processing. This is in addition to warnings and errors. This is the default value.

‘verbose’

Same as info, except more verbose.

‘debug’

Show everything, including debugging information.

By default the program logs to stderr, if coloring is supported by the terminal, colors are used to mark errors and warnings. Log coloring can be disabled setting the environment variable AV_LOG_FORCE_NOCOLOR or NO_COLOR, or can be forced setting the environment variable AV_LOG_FORCE_COLOR. The use of the environment variable NO_COLOR is deprecated and will be dropped in a following FFmpeg version.

‘-report’

Dump full command line and console output to a file named *program*-*YYYYMMDD*-*HHMMSS*.log in the current directory. This file can be useful for bug reports. It also implies -loglevel verbose.

Setting the environment variable FFREPORT to any value has the same effect. If the value is a ’:’-separated key=value sequence, these options will affect the report; options values must be escaped if they contain special characters or the options delimiter ’:’ (see the “Quoting and escaping” section in the ffmpeg-utils manual). The following option is recognized:

‘file’

set the file name to use for the report; %p is expanded to the name of the program, %t is expanded to a timestamp, %% is expanded to a plain %

Errors in parsing the environment variable are not fatal, and will not appear in the report.

‘-cpuflags flags (*global*)’

Allows setting and clearing cpu flags. This option is intended for testing. Do not use it unless you know what you’re doing.

```
ffmpeg -cpuflags -sse+mmx ...
ffmpeg -cpuflags mmx ...
ffmpeg -cpuflags 0 ...
```

Possible flags for this option are:

‘x86’

‘mmx’
‘mmxext’
‘sse’
‘sse2’
‘sse2slow’
‘sse3’
‘sse3slow’
‘ssse3’
‘atom’
‘sse4.1’
‘sse4.2’
‘avx’
‘xop’
‘fma4’
‘3dnow’
‘3dnowext’
‘cmov’

‘ARM’

‘armv5te’
‘armv6’
‘armv6t2’
‘vfp’

‘vfpv3’
‘neon’

‘PowerPC’

‘altivec’

‘Specific Processors’

‘pentium2’
‘pentium3’
‘pentium4’
‘k6’
‘k62’
‘athlon’
‘athlonxp’
‘k8’

‘-opencl_options options (*global*)’

Set OpenCL environment options. This option is only available when FFmpeg has been compiled with `--enable-opencl`.

options must be a list of *key=value* option pairs separated by ':'. See the “OpenCL Options” section in the ffmpeg-utils manual for the list of supported options.

5.3 AVOptions

These options are provided directly by the libavformat, libavdevice and libavcodec libraries. To see the list of available AVOptions, use the ‘`-help`’ option. They are separated into two categories:

‘generic’

These options can be set for any container, codec or device. Generic options are listed under AVFormatContext options for containers/devices and under AVCodecContext options for codecs.

‘private’

These options are specific to the given container, device or codec. Private options are listed under their corresponding containers/devices/codecs.

For example to write an ID3v2.3 header instead of a default ID3v2.4 to an MP3 file, use the ‘`id3v2_version`’ private option of the MP3 muxer:

```
ffmpeg -i input.flac -id3v2_version 3 out.mp3
```

All codec AVOptions are per-stream, and thus a stream specifier should be attached to them.

Note: the ‘`-nooption`’ syntax cannot be used for boolean AVOptions, use ‘`-option 0`’/‘`-option 1`’.

Note: the old undocumented way of specifying per-stream AVOptions by prepending v/a/s to the options name is now obsolete and will be removed soon.

5.4 Main options

‘`-f` *fmt* (*input/output*)’

Force input or output file format. The format is normally auto detected for input files and guessed from the file extension for output files, so this option is not needed in most cases.

‘`-i` *filename* (*input*)’

input file name

‘`-y` (*global*)’

Overwrite output files without asking.

‘`-n` (*global*)’

Do not overwrite output files, and exit immediately if a specified output file already exists.

‘-c[:*stream_specifier*] *codec* (*input/output*,*per-stream*)’
‘-codec[:*stream_specifier*] *codec* (*input/output*,*per-stream*)’

Select an encoder (when used before an output file) or a decoder (when used before an input file) for one or more streams. *codec* is the name of a decoder/encoder or a special value `copy` (output only) to indicate that the stream is not to be re-encoded.

For example

```
ffmpeg -i INPUT -map 0 -c:v libx264 -c:a copy OUTPUT
```

encodes all video streams with libx264 and copies all audio streams.

For each stream, the last matching `c` option is applied, so

```
ffmpeg -i INPUT -map 0 -c copy -c:v:1 libx264 -c:a:137 libvorbis OUTPUT
```

will copy all the streams except the second video, which will be encoded with libx264, and the 138th audio, which will be encoded with libvorbis.

‘-t *duration* (*output*)’

Stop writing the output after its duration reaches *duration*. *duration* may be a number in seconds, or in `hh:mm:ss[.xxx]` form.

-to and -t are mutually exclusive and -t has priority.

‘-to *position* (*output*)’

Stop writing the output at *position*. *position* may be a number in seconds, or in `hh:mm:ss[.xxx]` form.

-to and -t are mutually exclusive and -t has priority.

‘-fs *limit_size* (*output*)’

Set the file size limit, expressed in bytes.

‘-ss *position* (*input/output*)’

When used as an input option (before `-i`), seeks in this input file to *position*. Note the in most formats it is not possible to seek exactly, so `ffmpeg` will seek to the closest seek point before *position*. When transcoding and ‘`-accurate_seek`’ is enabled (the default), this extra segment between the seek point and *position* will be decoded and discarded. When doing stream copy or when ‘`-noaccurate_seek`’ is used, it will be preserved.

When used as an output option (before an output filename), decodes but discards input until the timestamps reach *position*.

position may be either in seconds or in `hh:mm:ss[.xxx]` form.

‘-itsoffset *offset* (*input*)’

Set the input time offset in seconds. `[-]hh:mm:ss[.xxx]` syntax is also supported. The offset is added to the timestamps of the input files. Specifying a positive offset means that the corresponding streams are delayed by *offset* seconds.

‘-timestamp *time* (*output*)’

Set the recording timestamp in the container. The syntax for *time* is:

```
now|([(YYYY-MM-DD|YYYYMMDD)[T|t| ]](HH:MM:SS[.m...])|(HHMMSS[.m...]))[Z|z])
```

If the value is "now" it takes the current time. Time is local time unless ’Z’ or ’z’ is appended, in which case it is interpreted as UTC. If the year-month-day part is not specified it takes the current year-month-day.

‘-metadata[:metadata_specifier] *key*=*value* (*output*,*per-metadata*)’

Set a metadata key/value pair.

An optional *metadata_specifier* may be given to set metadata on streams or chapters. See `-map_metadata` documentation for details.

This option overrides metadata set with `-map_metadata`. It is also possible to delete metadata by using an empty value.

For example, for setting the title in the output file:

```
ffmpeg -i in.avi -metadata title="my title" out.flv
```

To set the language of the first audio stream:

```
ffmpeg -i INPUT -metadata:s:a:1 language=eng OUTPUT
```

‘`-target type (output)`’

Specify target file type (`vcd`, `svcd`, `dvd`, `dv`, `dv50`). *type* may be prefixed with `pal-`, `ntsc-` or `film-` to use the corresponding standard. All the format options (bitrate, codecs, buffer sizes) are then set automatically. You can just type:

```
ffmpeg -i myfile.avi -target vcd /tmp/vcd.mpg
```

Nevertheless you can specify additional options as long as you know they do not conflict with the standard, as in:

```
ffmpeg -i myfile.avi -target vcd -bf 2 /tmp/vcd.mpg
```

‘`-dframes number (output)`’

Set the number of data frames to record. This is an alias for `-frames:d`.

‘`-frames[:stream_specifier] framecount (output,per-stream)`’

Stop writing to the stream after *framecount* frames.

‘`-q[:stream_specifier] q (output,per-stream)`’
‘`-qscale[:stream_specifier] q (output,per-stream)`’

Use fixed quality scale (VBR). The meaning of *q*/*qscale* is codec-dependent. If *qscale* is used without a *stream_specifier* then it applies only to the video stream, this is to maintain compatibility with previous behavior and as specifying the same codec specific value to 2 different codecs that is audio and video generally is not what is intended when no stream_specifier is used.

‘`-filter[:stream_specifier] filtergraph (output,per-stream)`’

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

filtergraph is a description of the filtergraph to apply to the stream, and must have a single input and a single output of the same type of the stream. In the filtergraph, the input is associated to the label `in`, and the output to the label `out`. See the ffmpeg-filters manual for more information about the filtergraph syntax.

See the -filter_complex option if you want to create filtergraphs with multiple inputs and/or outputs.

‘`-filter_script[:stream_specifier] filename (output,per-stream)`’

This option is similar to ‘`-filter`’, the only difference is that its argument is the name of the file from which a filtergraph description is to be read.

‘`-pre[:stream_specifier] preset_name (output,per-stream)`’

Specify the preset for matching stream(s).

‘`-stats (global)`’

Print encoding progress/statistics. It is on by default, to explicitly disable it you need to specify `-nostats`.

‘`-progress url (global)`’

Send program-friendly progress information to *url*.

Progress information is written approximately every second and at the end of the encoding process. It is made of "*key*=*value*" lines. *key* consists of only alphanumeric characters. The last key of a sequence of progress information is always "progress".

‘`-stdin`’

Enable interaction on standard input. On by default unless standard input is used as an input. To explicitly disable interaction you need to specify `-nostdin`.

Disabling interaction on standard input is useful, for example, if ffmpeg is in the background process group. Roughly the same result can be achieved with `ffmpeg ... < /dev/null` but it requires a shell.

‘-debug_ts (*global*)’

Print timestamp information. It is off by default. This option is mostly useful for testing and debugging purposes, and the output format may change from one version to another, so it should not be employed by portable scripts.

See also the option -fdebug ts.

‘-attach *filename* (*output*)’

Add an attachment to the output file. This is supported by a few formats like Matroska for e.g. fonts used in rendering subtitles. Attachments are implemented as a specific type of stream, so this option will add a new stream to the file. It is then possible to use per-stream options on this stream in the usual way. Attachment streams created with this option will be created after all the other streams (i.e. those created with -map or automatic mappings).

Note that for Matroska you also have to set the mimetype metadata tag:

```
ffmpeg -i INPUT -attach DejaVuSans.ttf -metadata:s:2 mimetype=application/x-truetype-font out.mkv
```

(assuming that the attachment stream will be third in the output file).

‘-dump_attachment[:*stream_specifier*] *filename* (*input,per-stream*)’

Extract the matching attachment stream into a file named *filename*. If *filename* is empty, then the value of the filename metadata tag will be used.

E.g. to extract the first attachment to a file named ’out.ttf’:

```
ffmpeg -dump_attachment:t:0 out.ttf -i INPUT
```

To extract all attachments to files determined by the filename tag:

```
ffmpeg -dump_attachment:t "" -i INPUT
```

Technical note – attachments are implemented as codec extradata, so this option can actually be used to extract extradata from any stream, not just attachments.

5.5 Video Options

‘-vframes *number* (*output*)’

Set the number of video frames to record. This is an alias for -frames:v.

‘-r[:*stream_specifier*] *fps* (*input/output,per-stream*)’

Set frame rate (Hz value, fraction or abbreviation).

As an input option, ignore any timestamps stored in the file and instead generate timestamps assuming constant frame rate *fps*.

As an output option, duplicate or drop input frames to achieve constant output frame rate *fps*.

‘-s[:*stream_specifier*] *size* (*input/output,per-stream*)’

Set frame size.

As an input option, this is a shortcut for the ‘video_size’ private option, recognized by some demuxers for which the frame size is either not stored in the file or is configurable – e.g. raw video or video grabbers.

As an output option, this inserts the scale video filter to the *end* of the corresponding filtergraph. Please use the scale filter directly to insert it at the beginning or some other place.

The format is ‘wxh’ (default - same as source).

‘-aspect[:*stream_specifier*] *aspect* (*output,per-stream*)’

Set the video display aspect ratio specified by *aspect*.

aspect can be a floating point number string, or a string of the form *num*:*den*, where *num* and *den* are the numerator and denominator of the aspect ratio. For example "4:3", "16:9", "1.3333", and "1.7777" are valid argument values.

If used together with '`-vcodec copy`', it will affect the aspect ratio stored at container level, but not the aspect ratio stored in encoded frames, if it exists.

'`-vn (output)`'

Disable video recording.

'`-vcodec codec (output)`'

Set the video codec. This is an alias for `-codec:v`.

'`-pass[:stream_specifier] n (output,per-stream)`'

Select the pass number (1 or 2). It is used to do two-pass video encoding. The statistics of the video are recorded in the first pass into a log file (see also the option -passlogfile), and in the second pass that log file is used to generate the video at the exact requested bitrate. On pass 1, you may just deactivate audio and set output to null, examples for Windows and Unix:

```
ffmpeg -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y NUL
ffmpeg -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y /dev/null
```

'`-passlogfile[:stream_specifier] prefix (output,per-stream)`'

Set two-pass log file name prefix to *prefix*, the default file name prefix is "ffmpeg2pass". The complete file name will be '`PREFIX-N.log`', where N is a number specific to the output stream

'`-vlang code`'

Set the ISO 639 language code (3 letters) of the current video stream.

'`-vf filtergraph (output)`'

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

This is an alias for `-filter:v`, see the -filter option.

5.6 Advanced Video Options

'`-pix_fmt[:stream_specifier] format (input/output,per-stream)`'

Set pixel format. Use `-pix_fmts` to show all the supported pixel formats. If the selected pixel format can not be selected, ffmpeg will print a warning and select the best pixel format supported by the encoder. If *pix_fmt* is prefixed by a `+`, ffmpeg will exit with an error if the requested pixel format can not be selected, and automatic conversions inside filtergraphs are disabled. If *pix_fmt* is a single `+`, ffmpeg selects the same pixel format as the input (or graph output) and automatic conversions are disabled.

'`-sws_flags flags (input/output)`'

Set SwScaler flags.

'`-vdt n`'

Discard threshold.

'`-rc_override[:stream_specifier] override (output,per-stream)`'

Rate control override for specific intervals, formatted as "int,int,int" list separated with slashes. Two first values are the beginning and end frame numbers, last one is quantizer to use if positive, or quality factor if negative.

'`-ilme`'

Force interlacing support in encoder (MPEG-2 and MPEG-4 only). Use this option if your input file is interlaced and you want to keep the interlaced format for minimum losses. The alternative is to deinterlace the input stream with '`-deinterlace`', but deinterlacing introduces losses.

'`-psnr`'

Calculate PSNR of compressed frames.

'`-vstats`'

Dump video coding statistics to '`vstats_HHMMSS.log`'.

'`-vstats_file file`'

Dump video coding statistics to *file*.

‘-top[:*stream_specifier*] *n* (*output*,*per-stream*)’

top=1/bottom=0/auto=-1 field first

‘-dc *precision*’

Intra_dc_precision.

‘-vtag *fourcc/tag* (*output*)’

Force video tag/fourcc. This is an alias for -tag:v.

‘-qphist (*global*)’

Show QP histogram

‘-vbsf *bitstream_filter*’

Deprecated see -bsf

‘-force_key_frames[:*stream_specifier*] *time*[,*time*...] (*output*,*per-stream*)’
‘-force_key_frames[:*stream_specifier*] expr:*expr* (*output*,*per-stream*)’

Force key frames at the specified timestamps, more precisely at the first frames after each specified time.

If the argument is prefixed with expr:, the string *expr* is interpreted like an expression and is evaluated for each frame. A key frame is forced in case the evaluation is non-zero.

If one of the times is "chapters[*delta*]", it is expanded into the time of the beginning of all chapters in the file, shifted by *delta*, expressed as a time in seconds. This option can be useful to ensure that a seek point is present at a chapter mark or any other designated place in the output file.

For example, to insert a key frame at 5 minutes, plus key frames 0.1 second before the beginning of every chapter:

```
-force_key_frames 0:05:00,chapters-0.1
```

The expression in *expr* can contain the following constants:

‘n’

the number of current processed frame, starting from 0

‘n_forced’

the number of forced frames

‘prev_forced_n’

the number of the previous forced frame, it is NAN when no keyframe was forced yet

‘prev_forced_t’

the time of the previous forced frame, it is NAN when no keyframe was forced yet

‘t’

the time of the current processed frame

For example to force a key frame every 5 seconds, you can specify:

```
-force_key_frames expr:gte(t,n_forced*5)
```

To force a key frame 5 seconds after the time of the last forced one, starting from second 13:

```
-force_key_frames expr:if(isnan(prev_forced_t),gte(t,13),gte(t,prev_forced_t+5))
```

Note that forcing too many keyframes is very harmful for the lookahead algorithms of certain encoders: using fixed-GOP options or similar would be more efficient.

‘-copyinkf[:*stream_specifier*] (*output*,*per-stream*)’

When doing stream copy, copy also non-key frames found at the beginning.

‘-hwaccel[:*stream_specifier*] *hwaccel* (*input*,*per-stream*)’

Use hardware acceleration to decode the matching stream(s). The allowed values of *hwaccel* are:

‘none’

Do not use any hardware acceleration (the default).

‘auto’

Automatically select the hardware acceleration method.

‘vdpau’

Use VDPAU (Video Decode and Presentation API for Unix) hardware acceleration.

This option has no effect if the selected hwaccel is not available or not supported by the chosen decoder.

Note that most acceleration methods are intended for playback and will not be faster than software decoding on modern CPUs. Additionally, ffmpeg will usually need to copy the decoded frames from the GPU memory into the system memory, resulting in further performance loss. This option is thus mainly useful for testing.

‘-hwaccel_device[:*stream_specifier*] *hwaccel_device* (*input*,*per-stream*)’

Select a device to use for hardware acceleration.

This option only makes sense when the ‘-hwaccel’ option is also specified. Its exact meaning depends on the specific hardware acceleration method chosen.

‘vdpau’

For VDPAU, this option specifies the X11 display/screen to use. If this option is not specified, the value of the *DISPLAY* environment variable is used

5.7 Audio Options

‘-aframes *number* (*output*)’

Set the number of audio frames to record. This is an alias for -frames:a.

‘-ar[:*stream_specifier*] *freq* (*input/output*,*per-stream*)’

Set the audio sampling frequency. For output streams it is set by default to the frequency of the corresponding input stream. For input streams this option only makes sense for audio grabbing devices and raw demuxers and is mapped to the corresponding demuxer options.

‘-aq *q* (*output*)’

Set the audio quality (codec-specific, VBR). This is an alias for -q:a.

‘-ac[:*stream_specifier*] *channels* (*input/output*,*per-stream*)’

Set the number of audio channels. For output streams it is set by default to the number of input audio channels. For input streams this option only makes sense for audio grabbing devices and raw demuxers and is mapped to the corresponding demuxer options.

‘-an (*output*)’

Disable audio recording.

‘-acodec *codec* (*input/output*)’

Set the audio codec. This is an alias for -codec:a.

‘-sample_fmt[:*stream_specifier*] *sample_fmt* (*output*,*per-stream*)’

Set the audio sample format. Use -sample_fmts to get a list of supported sample formats.

‘-af *filtergraph* (*output*)’

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

This is an alias for `-filter:a`, see the -filter option.

5.8 Advanced Audio options:

'`-atag` *fourcc/tag* (*output*)'

Force audio tag/fourcc. This is an alias for `-tag:a`.

'`-absf` *bitstream_filter*'

Deprecated, see -bsf

'`-guess_layout_max` *channels* (*input,per-stream*)'

If some input channel layout is not known, try to guess only if it corresponds to at most the specified number of channels. For example, 2 tells to `ffmpeg` to recognize 1 channel as mono and 2 channels as stereo but not 6 channels as 5.1. The default is to always try to guess. Use 0 to disable all guessing.

5.9 Subtitle options:

'`-slang` *code*'

Set the ISO 639 language code (3 letters) of the current subtitle stream.

'`-scodec` *codec* (*input/output*)'

Set the subtitle codec. This is an alias for `-codec:s`.

'`-sn` (*output*)'

Disable subtitle recording.

'`-sbsf` *bitstream_filter*'

Deprecated, see -bsf

5.10 Advanced Subtitle options:

'`-fix_sub_duration`'

Fix subtitles durations. For each subtitle, wait for the next packet in the same stream and adjust the duration of the first to avoid overlap. This is necessary with some subtitles codecs, especially DVB subtitles, because the duration in the original packet is only a rough estimate and the end is actually marked by an empty subtitle frame. Failing to use this option when necessary can result in exaggerated durations or muxing failures due to non-monotonic timestamps.

Note that this option will delay the output of all data until the next subtitle packet is decoded: it may increase memory consumption and latency a lot.

'`-canvas_size` *size*'

Set the size of the canvas used to render subtitles.

5.11 Advanced options

'`-map` [`-`]*input_file_id*[`:`*stream_specifier*][`,`*sync_file_id*[`:`*stream_specifier*]] | *[linklabel]* (*output*)'

Designate one or more input streams as a source for the output file. Each input stream is identified by the input file index *input_file_id* and the input stream index *input_stream_id* within the input file. Both indices start at 0. If specified, *sync_file_id*:*stream_specifier* sets which input stream is used as a presentation sync reference.

The first `-map` option on the command line specifies the source for output stream 0, the second `-map` option specifies the source for output stream 1, etc.

A `-` character before the stream identifier creates a "negative" mapping. It disables matching streams from already created mappings.

An alternative *[linklabel]* form will map outputs from complex filter graphs (see the '`-filter_complex`' option) to the output file. *linklabel* must correspond to a defined output link label in the graph.

For example, to map ALL streams from the first input file to output

```
ffmpeg -i INPUT -map 0 output
```

For example, if you have two audio streams in the first input file, these streams are identified by "0:0" and "0:1". You can use `-map` to select which streams to place in an output file. For example:

```
ffmpeg -i INPUT -map 0:1 out.wav
```

will map the input stream in '`INPUT`' identified by "0:1" to the (single) output stream in '`out.wav`'.

For example, to select the stream with index 2 from input file '`a.mov`' (specified by the identifier "0:2"), and stream with index 6 from input '`b.mov`' (specified by the identifier "1:6"), and copy them to the output file '`out.mov`':

```
ffmpeg -i a.mov -i b.mov -c copy -map 0:2 -map 1:6 out.mov
```

To select all video and the third audio stream from an input file:

```
ffmpeg -i INPUT -map 0:v -map 0:a:2 OUTPUT
```

To map all the streams except the second audio, use negative mappings

```
ffmpeg -i INPUT -map 0 -map -0:a:1 OUTPUT
```

Note that using this option disables the default mappings for this output file.

'`-map_channel [`*input_file_id.stream_specifier.channel_id*`|-1][:`*output_file_id.stream_specifier*`]`'

Map an audio channel from a given input to an output. If *output_file_id.stream_specifier* is not set, the audio channel will be mapped on all the audio streams.

Using "-1" instead of *input_file_id.stream_specifier.channel_id* will map a muted channel.

For example, assuming *INPUT* is a stereo audio file, you can switch the two audio channels with the following command:

```
ffmpeg -i INPUT -map_channel 0.0.1 -map_channel 0.0.0 OUTPUT
```

If you want to mute the first channel and keep the second:

```
ffmpeg -i INPUT -map_channel -1 -map_channel 0.0.1 OUTPUT
```

The order of the "-map_channel" option specifies the order of the channels in the output stream. The output channel layout is guessed from the number of channels mapped (mono if one "-map_channel", stereo if two, etc.). Using "-ac" in combination of "-map_channel" makes the channel gain levels to be updated if input and output channel layouts don't match (for instance two "-map_channel" options and "-ac 6").

You can also extract each channel of an input to specific outputs; the following command extracts two channels of the *INPUT* audio stream (file 0, stream 0) to the respective *OUTPUT_CH0* and *OUTPUT_CH1* outputs:

```
ffmpeg -i INPUT -map_channel 0.0.0 OUTPUT_CH0 -map_channel 0.0.1 OUTPUT_CH1
```

The following example splits the channels of a stereo input into two separate streams, which are put into the same output file:

```
ffmpeg -i stereo.wav -map 0:0 -map 0:0 -map_channel 0.0.0:0.0 -map_channel 0.0.1:0.1 -y out.ogg
```

Note that currently each output stream can only contain channels from a single input stream; you can't for example use "-map_channel" to pick multiple input audio channels contained in different streams (from the same or different files) and merge them into a single output stream. It is therefore not currently possible, for example, to turn two separate mono streams into a single stereo stream. However splitting a stereo stream into two single channel mono streams is possible.

If you need this feature, a possible workaround is to use the *amerge* filter. For example, if you need to merge a media (here

‘input.mkv’) with 2 mono audio streams into one single stereo channel audio stream (and keep the video stream), you can use the following command:

```
ffmpeg -i input.mkv -filter_complex "[0:1] [0:2] amerge" -c:a pcm_s16le -c:v copy output.mkv
```

‘-map_metadata[:*metadata_spec_out*] *infile*[:*metadata_spec_in*] (*output*,*per-metadata*)’

Set metadata information of the next output file from *infile*. Note that those are file indices (zero-based), not filenames. Optional *metadata_spec_in/out* parameters specify, which metadata to copy. A metadata specifier can have the following forms:

‘*g*’

global metadata, i.e. metadata that applies to the whole file

‘*s*[:*stream_spec*]’

per-stream metadata. *stream_spec* is a stream specifier as described in the Stream specifiers chapter. In an input metadata specifier, the first matching stream is copied from. In an output metadata specifier, all matching streams are copied to.

‘*c*:*chapter_index*’

per-chapter metadata. *chapter_index* is the zero-based chapter index.

‘*p*:*program_index*’

per-program metadata. *program_index* is the zero-based program index.

If metadata specifier is omitted, it defaults to global.

By default, global metadata is copied from the first input file, per-stream and per-chapter metadata is copied along with streams/chapters. These default mappings are disabled by creating any mapping of the relevant type. A negative file index can be used to create a dummy mapping that just disables automatic copying.

For example to copy metadata from the first stream of the input file to global metadata of the output file:

```
ffmpeg -i in.ogg -map_metadata 0:s:0 out.mp3
```

To do the reverse, i.e. copy global metadata to all audio streams:

```
ffmpeg -i in.mkv -map_metadata:s:a 0:g out.mkv
```

Note that simple `0` would work as well in this example, since global metadata is assumed by default.

‘-map_chapters *input_file_index* (*output*)’

Copy chapters from input file with index *input_file_index* to the next output file. If no chapter mapping is specified, then chapters are copied from the first input file with at least one chapter. Use a negative file index to disable any chapter copying.

‘-benchmark (*global*)’

Show benchmarking information at the end of an encode. Shows CPU time used and maximum memory consumption. Maximum memory consumption is not supported on all systems, it will usually display as 0 if not supported.

‘-benchmark_all (*global*)’

Show benchmarking information during the encode. Shows CPU time used in various steps (audio/video encode/decode).

‘-timelimit *duration* (*global*)’

Exit after ffmpeg has been running for *duration* seconds.

‘-dump (*global*)’

Dump each input packet to stderr.

‘-hex (*global*)’

When dumping packets, also dump the payload.

‘-re (*input*)’

Read input at native frame rate. Mainly used to simulate a grab device. or live input stream (e.g. when reading from a file). Should not be used with actual grab devices or live input streams (where it can cause packet loss). By default `ffmpeg` attempts to read the input(s) as fast as possible. This option will slow down the reading of the input(s) to the native frame rate of the input(s). It is useful for real-time output (e.g. live streaming).

'`-loop_input`'

Loop over the input stream. Currently it works only for image streams. This option is used for automatic FFserver testing. This option is deprecated, use -loop 1.

'`-loop_output` *`number_of_times`*'

Repeatedly loop output for formats that support looping such as animated GIF (0 will loop the output infinitely). This option is deprecated, use -loop.

'`-vsync` *`parameter`*'

Video sync method. For compatibility reasons old values can be specified as numbers. Newly added values will have to be specified as strings always.

'`0, passthrough`'

Each frame is passed with its timestamp from the demuxer to the muxer.

'`1, cfr`'

Frames will be duplicated and dropped to achieve exactly the requested constant frame rate.

'`2, vfr`'

Frames are passed through with their timestamp or dropped so as to prevent 2 frames from having the same timestamp.

'`drop`'

As passthrough but destroys all timestamps, making the muxer generate fresh timestamps based on frame-rate.

'`-1, auto`'

Chooses between 1 and 2 depending on muxer capabilities. This is the default method.

Note that the timestamps may be further modified by the muxer, after this. For example, in the case that the format option '`avoid_negative_ts`' is enabled.

With -map you can select from which stream the timestamps should be taken. You can leave either video or audio unchanged and sync the remaining stream(s) to the unchanged one.

'`-async` *`samples_per_second`*'

Audio sync method. "Stretches/squeezes" the audio stream to match the timestamps, the parameter is the maximum samples per second by which the audio is changed. -async 1 is a special case where only the start of the audio stream is corrected without any later correction.

Note that the timestamps may be further modified by the muxer, after this. For example, in the case that the format option '`avoid_negative_ts`' is enabled.

This option has been deprecated. Use the `aresample` audio filter instead.

'`-copyts`'

Do not process input timestamps, but keep their values without trying to sanitize them. In particular, do not remove the initial start time offset value.

Note that, depending on the '`vsync`' option or on specific muxer processing (e.g. in case the format option '`avoid_negative_ts`' is enabled) the output timestamps may mismatch with the input timestamps even when this option is selected.

'`-copytb` *`mode`*'

Specify how to set the encoder timebase when stream copying. *mode* is an integer numeric value, and can assume one of the following values:

'`1`'

Use the demuxer timebase.

The time base is copied to the output encoder from the corresponding input demuxer. This is sometimes required to avoid

non monotonically increasing timestamps when copying video streams with variable frame rate.

‘0’

Use the decoder timebase.

The time base is copied to the output encoder from the corresponding input decoder.

‘-1’

Try to make the choice automatically, in order to generate a sane output.

Default value is -1.

‘-shortest (*output*)’

Finish encoding when the shortest input stream ends.

‘-dts_delta_threshold’

Timestamp discontinuity delta threshold.

‘-muxdelay *seconds* (*input*)’

Set the maximum demux-decode delay.

‘-muxpreload *seconds* (*input*)’

Set the initial demux-decode delay.

‘-streamid *output-stream-index*:*new-value* (*output*)’

Assign a new stream-id value to an output stream. This option should be specified prior to the output filename to which it applies. For the situation where multiple output files exist, a streamid may be reassigned to a different value.

For example, to set the stream 0 PID to 33 and the stream 1 PID to 36 for an output mpegts file:

```
ffmpeg -i infile -streamid 0:33 -streamid 1:36 out.ts
```

‘-bsf[:*stream_specifier*] *bitstream_filters* (*output*,*per-stream*)’

Set bitstream filters for matching streams. *bitstream_filters* is a comma-separated list of bitstream filters. Use the -bsfs option to get the list of bitstream filters.

```
ffmpeg -i h264.mp4 -c:v copy -bsf:v h264_mp4toannexb -an out.h264
```

```
ffmpeg -i file.mov -an -vn -bsf:s mov2textsub -c:s copy -f rawvideo sub.txt
```

‘-tag[:*stream_specifier*] *codec_tag* (*per-stream*)’

Force a tag/fourcc for matching streams.

‘-timecode *hh*:*mm*:*ss*SEP*ff*’

Specify Timecode for writing. *SEP* is ’:’ for non drop timecode and ’;’ (or ’.’) for drop.

```
ffmpeg -i input.mpg -timecode 01:02:03.04 -r 30000/1001 -s ntsc output.mpg
```

‘-filter_complex *filtergraph* (*global*)’

Define a complex filtergraph, i.e. one with arbitrary number of inputs and/or outputs. For simple graphs – those with one input and one output of the same type – see the ‘-filter’ options. *filtergraph* is a description of the filtergraph, as described in the “Filtergraph syntax” section of the ffmpeg-filters manual.

Input link labels must refer to input streams using the [file_index:stream_specifier] syntax (i.e. the same as ‘-map’ uses). If *stream_specifier* matches multiple streams, the first one will be used. An unlabeled input will be connected to the first unused input stream of the matching type.

Output link labels are referred to with ‘-map’. Unlabeled outputs are added to the first output file.

Note that with this option it is possible to use only lavfi sources without normal input files.

For example, to overlay an image over video

```
ffmpeg -i video.mkv -i image.png -filter_complex '[0:v][1:v]overlay[out]' -map
'[out]' out.mkv
```

Here `[0:v]` refers to the first video stream in the first input file, which is linked to the first (main) input of the overlay filter. Similarly the first video stream in the second input is linked to the second (overlay) input of overlay.

Assuming there is only one video stream in each input file, we can omit input labels, so the above is equivalent to

```
ffmpeg -i video.mkv -i image.png -filter_complex 'overlay[out]' -map
'[out]' out.mkv
```

Furthermore we can omit the output label and the single output from the filter graph will be added to the output file automatically, so we can simply write

```
ffmpeg -i video.mkv -i image.png -filter_complex 'overlay' out.mkv
```

To generate 5 seconds of pure red video using lavfi `color` source:

```
ffmpeg -filter_complex 'color=c=red' -t 5 out.mkv
```

'`-lavfi` *filtergraph* (*global*)'

Define a complex filtergraph, i.e. one with arbitrary number of inputs and/or outputs. Equivalent to '`-filter_complex`'.

'`-filter_complex_script` *filename* (*global*)'

This option is similar to '`-filter_complex`', the only difference is that its argument is the name of the file from which a complex filtergraph description is to be read.

'`-accurate_seek` (*input*)'

This option enables or disables accurate seeking in input files with the '`-ss`' option. It is enabled by default, so seeking is accurate when transcoding. Use '`-noaccurate_seek`' to disable it, which may be useful e.g. when copying some streams and transcoding the others.

'`-override_ffserver` (*global*)'

Overrides the input specifications from `ffserver`. Using this option you can map any input stream to `ffserver` and control many aspects of the encoding from `ffmpeg`. Without this option `ffmpeg` will transmit to `ffserver` what is requested by `ffserver`.

The option is intended for cases where features are needed that cannot be specified to `ffserver` but can be to `ffmpeg`.

As a special exception, you can use a bitmap subtitle stream as input: it will be converted into a video with the same size as the largest video in the file, or 720x576 if no video is present. Note that this is an experimental and temporary solution. It will be removed once libavfilter has proper support for subtitles.

For example, to hardcode subtitles on top of a DVB-T recording stored in MPEG-TS format, delaying the subtitles by 1 second:

```
ffmpeg -i input.ts -filter_complex \
  '[#0x2ef] setpts=PTS+1/TB [sub] ; [#0x2d0] [sub] overlay' \
  -sn -map '#0x2dc' output.mkv
```

(0x2d0, 0x2dc and 0x2ef are the MPEG-TS PIDs of respectively the video, audio and subtitles streams; 0:0, 0:3 and 0:7 would have worked too)

5.12 Preset files

A preset file contains a sequence of *option=value* pairs, one for each line, specifying a sequence of options which would be awkward to specify on the command line. Lines starting with the hash ('#') character are ignored and are used to provide comments. Check the '`presets`' directory in the FFmpeg source tree for examples.

Preset files are specified with the `vpre`, `apre`, `spre`, and `fpre` options. The `fpre` option takes the filename of the preset instead of a preset name as input and can be used for any kind of codec. For the `vpre`, `apre`, and `spre` options, the options specified in a preset file are applied to the currently selected codec of the same type as the preset option.

The argument passed to the `vpre`, `apre`, and `spre` preset options identifies the preset file to use according to the following rules:

First ffmpeg searches for a file named *arg*.ffpreset in the directories '`$FFMPEG_DATADIR`' (if set), and '`$HOME/.ffmpeg`', and in the datadir defined at configuration time (usually '`PREFIX/share/ffmpeg`') or in a '`ffpresets`' folder along the executable on win32, in that order. For example, if the argument is `libvpx-1080p`, it will search for the file '`libvpx-1080p.ffpreset`'.

If no such file is found, then ffmpeg will search for a file named *codec_name-arg*.ffpreset in the above-mentioned directories, where *codec_name* is the name of the codec to which the preset file options will be applied. For example, if you select the video codec with `-vcodec libvpx` and use `-vpre 1080p`, then it will search for the file '`libvpx-1080p.ffpreset`'.

6. Tips

- For streaming at very low bitrates, use a low frame rate and a small GOP size. This is especially true for RealVideo where the Linux player does not seem to be very fast, so it can miss frames. An example is:

```
ffmpeg -g 3 -r 3 -t 10 -b:v 50k -s qcif -f rv10 /tmp/b.rm
```

- The parameter 'q' which is displayed while encoding is the current quantizer. The value 1 indicates that a very good quality could be achieved. The value 31 indicates the worst quality. If q=31 appears too often, it means that the encoder cannot compress enough to meet your bitrate. You must either increase the bitrate, decrease the frame rate or decrease the frame size.
- If your computer is not fast enough, you can speed up the compression at the expense of the compression ratio. You can use '-me zero' to speed up motion estimation, and '-g 0' to disable motion estimation completely (you have only I-frames, which means it is about as good as JPEG compression).
- To have very low audio bitrates, reduce the sampling frequency (down to 22050 Hz for MPEG audio, 22050 or 11025 for AC-3).
- To have a constant quality (but a variable bitrate), use the option '-qscale n' when 'n' is between 1 (excellent quality) and 31 (worst quality).

7. Examples

7.1 Preset files

A preset file contains a sequence of *option=value* pairs, one for each line, specifying a sequence of options which can be specified also on the command line. Lines starting with the hash ('#') character are ignored and are used to provide comments. Empty lines are also ignored. Check the '`presets`' directory in the FFmpeg source tree for examples.

Preset files are specified with the `pre` option, this option takes a preset name as input. FFmpeg searches for a file named *preset_name*.avpreset in the directories '`$AVCONV_DATADIR`' (if set), and '`$HOME/.ffmpeg`', and in the data directory defined at configuration time (usually '`$PREFIX/share/ffmpeg`') in that order. For example, if the argument is `libx264-max`, it will search for the file '`libx264-max.avpreset`'.

7.2 Video and Audio grabbing

If you specify the input format and device then ffmpeg can grab video and audio directly.

```
ffmpeg -f oss -i /dev/dsp -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

Or with an ALSA audio source (mono input, card id 1) instead of OSS:

```
ffmpeg -f alsa -ac 1 -i hw:1 -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

Note that you must activate the right video source and channel before launching ffmpeg with any TV viewer such as xawtv by Gerd Knorr. You also have to set the audio recording levels correctly with a standard mixer.

7.3 X11 grabbing

Grab the X11 display with ffmpeg via

```
ffmpeg -f x11grab -video_size cif -framerate 25 -i :0.0 /tmp/out.mpg
```

0.0 is display.screen number of your X11 server, same as the DISPLAY environment variable.

```
ffmpeg -f x11grab -video_size cif -framerate 25 -i :0.0+10,20 /tmp/out.mpg
```

0.0 is display.screen number of your X11 server, same as the DISPLAY environment variable. 10 is the x-offset and 20 the y-offset for the grabbing.

7.4 Video and Audio file format conversion

Any supported file format and protocol can serve as input to ffmpeg:

Examples:

- You can use YUV files as input:

  ```
  ffmpeg -i /tmp/test%d.Y /tmp/out.mpg
  ```

  It will use the files:

  ```
  /tmp/test0.Y, /tmp/test0.U, /tmp/test0.V,
  /tmp/test1.Y, /tmp/test1.U, /tmp/test1.V, etc...
  ```

  The Y files use twice the resolution of the U and V files. They are raw files, without header. They can be generated by all decent video decoders. You must specify the size of the image with the '`-s`' option if ffmpeg cannot guess it.

- You can input from a raw YUV420P file:

  ```
  ffmpeg -i /tmp/test.yuv /tmp/out.avi
  ```

  test.yuv is a file containing raw YUV planar data. Each frame is composed of the Y plane followed by the U and V planes at half vertical and horizontal resolution.

- You can output to a raw YUV420P file:

  ```
  ffmpeg -i mydivx.avi hugefile.yuv
  ```

- You can set several input files and output files:

  ```
  ffmpeg -i /tmp/a.wav -s 640x480 -i /tmp/a.yuv /tmp/a.mpg
  ```

  Converts the audio file a.wav and the raw YUV video file a.yuv to MPEG file a.mpg.

- You can also do audio and video conversions at the same time:

  ```
  ffmpeg -i /tmp/a.wav -ar 22050 /tmp/a.mp2
  ```

  Converts a.wav to MPEG audio at 22050 Hz sample rate.

- You can encode to several formats at the same time and define a mapping from input stream to output streams:

  ```
  ffmpeg -i /tmp/a.wav -map 0:a -b:a 64k /tmp/a.mp2 -map 0:a -b:a 128k /tmp/b.mp2
  ```

  Converts a.wav to a.mp2 at 64 kbits and to b.mp2 at 128 kbits. '-map file:index' specifies which input stream is used for each output stream, in the order of the definition of output streams.

- You can transcode decrypted VOBs:

  ```
  ffmpeg -i snatch_1.vob -f avi -c:v mpeg4 -b:v 800k -g 300 -bf 2 -c:a libmp3lame -b:a 128k snatch.avi
  ```

  This is a typical DVD ripping example; the input is a VOB file, the output an AVI file with MPEG-4 video and MP3 audio. Note that in this command we use B-frames so the MPEG-4 stream is DivX5 compatible, and GOP size is 300 which means one intra frame every 10 seconds for 29.97fps input video. Furthermore, the audio stream is MP3-encoded so you need to enable LAME support by passing `--enable-libmp3lame` to configure. The mapping is particularly useful for DVD transcoding to get the desired audio language.

  NOTE: To see the supported input formats, use `ffmpeg -formats`.

- You can extract images from a video, or create a video from many images:

  For extracting images from a video:

  ```
  ffmpeg -i foo.avi -r 1 -s WxH -f image2 foo-%03d.jpeg
  ```

This will extract one video frame per second from the video and will output them in files named ‘`foo-001.jpeg`’, ‘`foo-002.jpeg`’, etc. Images will be rescaled to fit the new WxH values.

If you want to extract just a limited number of frames, you can use the above command in combination with the -vframes or -t option, or in combination with -ss to start extracting from a certain point in time.

For creating a video from many images:

```
ffmpeg -f image2 -i foo-%03d.jpeg -r 12 -s WxH foo.avi
```

The syntax `foo-%03d.jpeg` specifies to use a decimal number composed of three digits padded with zeroes to express the sequence number. It is the same syntax supported by the C printf function, but only formats accepting a normal integer are suitable.

When importing an image sequence, -i also supports expanding shell-like wildcard patterns (globbing) internally, by selecting the image2-specific `-pattern_type glob` option.

For example, for creating a video from filenames matching the glob pattern `foo-*.jpeg`:

```
ffmpeg -f image2 -pattern_type glob -i 'foo-*.jpeg' -r 12 -s WxH foo.avi
```

- You can put many streams of the same type in the output:

```
ffmpeg -i test1.avi -i test2.avi -map 0:3 -map 0:2 -map 0:1 -map 0:0 -c copy test12.nut
```

The resulting output file ‘`test12.avi`’ will contain first four streams from the input file in reverse order.

- To force CBR video output:

```
ffmpeg -i myfile.avi -b 4000k -minrate 4000k -maxrate 4000k -bufsize 1835k out.m2v
```

- The four options lmin, lmax, mblmin and mblmax use ’lambda’ units, but you may use the QP2LAMBDA constant to easily convert from ’q’ units:

```
ffmpeg -i src.ext -lmax 21*QP2LAMBDA dst.ext
```

8. See Also

ffmpeg-all, ffplay, ffprobe, ffserver, ffmpeg-utils, ffmpeg-scaler, ffmpeg-resampler, ffmpeg-codecs, ffmpeg-bitstream-filters, ffmpeg-formats, ffmpeg-devices, ffmpeg-protocols, ffmpeg-filters

9. Authors

The FFmpeg developers.

For details about the authorship, see the Git history of the project (git://source.ffmpeg.org/ffmpeg), e.g. by typing the command `git log` in the FFmpeg source directory, or browsing the online repository at http://source.ffmpeg.org.

Maintainers for the specific components are listed in the file ‘`MAINTAINERS`’ in the source code tree.

This document was generated by *Kyle Schwarz* on *December 1, 2013* using *texi2html 1.82*.

[FFmpeg-user] Problem with setting ffprobe writer options for xml output

Tschöpel, Sebastian sebastian.tschoepel at iais.fraunhofer.de
Mon Jul 2 17:33:12 CEST 2012

- Previous message: [FFmpeg-user] YUV->RGB conversion. What is the default matrix used?
- Next message: [FFmpeg-user] Problem with setting ffprobe writer options for xml output
- **Messages sorted by:** [date] [thread] [subject] [author]

```
Hello List,

I am currently trying to use FFProbe to ouput stream information as XML.

When I use this command:

ffprobe -v quiet -print_format xml -show_streams sample.mp3

I get something like this:

<?xml version="1.0" encoding="UTF-8"?>
<ffprobe>
    <streams>
        <stream index="0" codec_name="mp3" codec_long_name="MP3 (MPEG audio layer 3)" codec_type="audio"
codec_time_base="1/44100" codec_tag_string="[0][0][0][0]" codec_tag="0x0000" sample_fmt="s16" sample_rate="44100"
channels="2" bits_per_sample="0" r_frame_rate="0/0" avg_frame_rate="0/0" time_base="1/14112000"
start_time="0.000000" duration="36.207687" bit_rate="128000"/>
    </streams>
</ffprobe>

According to the documentation it is also possible to get qualified and xsd compliant XML by passing the
arguments:

'fully_qualified, q'
'xsd_compliant, x'

to the writer.
My problem is: I have no idea how to set those parameters. Having given the hint from the documentation "-
print_format writer_name[=writer_options]" I tried various things such as:

-print_format:x=1 xml
-print_format xml:x=1
-print_format xml x=1
-print_foramt xml[x=1]
...

How to set those arguments correctly? Hope someone can give me some hints on this.

--
Best regards,
Basti
```

- Previous message: [FFmpeg-user] YUV->RGB conversion. What is the default matrix used?
- Next message: [FFmpeg-user] Problem with setting ffprobe writer options for xml output
- **Messages sorted by:** [date] [thread] [subject] [author]

XmlDocument.Load Method

.NET Framework 4.5 | Other Versions |
0 out of 3 rated this helpful - Rate this topic

Loads the specified XML data from a Stream, a URL, a TextReader, or an XmlReader.

This member is overloaded. For complete information about this member, including syntax, usage, and examples, click a name in the overload list.

Overload List

	Name	Description
	Load(Stream)	Loads the XML document from the specified stream.
	Load(String)	Loads the XML document from the specified URL.
	Load(TextReader)	Loads the XML document from the specified TextReader.
	Load(XmlReader)	Loads the XML document from the specified XmlReader.

Top

See Also

Reference
XmlDocument Class
System.Xml Namespace

[FFmpeg-user] analyzeduration and probesize

Tom Evans tevans.uk at googlemail.com
Thu Mar 28 12:48:12 CET 2013

- Previous message: [FFmpeg-user] analyzeduration and probesize
- Next message: [FFmpeg-user] av_read_frame
- **Messages sorted by:** [date] [thread] [subject] [author]

On Thu, Mar 28, 2013 at 4:54 AM, Young Kim <shadowing71 at gmail.com> wrote:
> *Hello,*
>
> *I've recently encountered something asking me to either increase analyzeduration or probesize in ffmpeg, but the documentation doesn't seem to be very clear at what either of these do. Can someone shed a light on what they do and what the differences are?*
>
> *Thanks,*
> *Young Kim*
>

```
ffmpeg -h full | grep 'analyzeduration\|probesize'

-probesize         <int>        .D.... set probing size (from 32 to INT_MAX)
-analyzeduration   <int>        .D.... specify how many microseconds
are analyzed to probe the input (from 0 to INT_MAX)
-fpsprobesize      <int>        .D.... number of frames used to probe
fps (from -1 to 2.14748e+09)

probesize is in bytes.

Cheers

Tom
```

- Previous message: [FFmpeg-user] analyzeduration and probesize
- Next message: [FFmpeg-user] av_read_frame
- **Messages sorted by:** [date] [thread] [subject] [author]

FFmpeg and x264 Encoding Guide

x264 is a H.264/MPEG-4 AVC encoder. The goal of this guide is to inform new users how to create a high-quality H.264 video.

There are two rate control modes that are usually suggested for general use: Constant Rate Factor (CRF) or Two-Pass ABR. The rate control is a method that will decide how many bits will be used for each frame. This will determine the file size and also how quality is distributed.

If you need help compiling and installing libx264 see one of our FFmpeg and x264 compiling guides.

Contents

Constant Rate Factor (CRF)

This method allows the encoder to attempt to achieve a certain output quality for the whole file when output file size is of less importance. This provides maximum compression efficiency with a single pass. Each frame gets the bitrate it needs to keep the requested quality level. The downside is that you can't tell it to get a specific filesize or not go over a specific size or bitrate.

1. Choose a CRF value

The range of the quantizer scale is 0-51: where 0 is lossless, 23 is default, and 51 is worst possible. A lower value is a higher quality and a subjectively sane range is 18-28. Consider 18 to be visually lossless or nearly so: it should look the same or nearly the same as the input but it isn't technically lossless.

The range is exponential, so increasing the CRF value +6 is roughly half the bitrate while -6 is roughly twice the bitrate. General usage is to choose the highest CRF value that still provides an acceptable quality. If the output looks good, then try a higher value and if it looks bad then choose a lower value.

> **Note:** The CRF quantizer scale mentioned on this page only applies to 8-bit x264 (10-bit x264 quantizer scale is 0-63). You can see what you are using with `x264 --help` listed under `Output bit depth`. 8-bit is more common among distributors.

2. Choose a preset

A preset is a collection of options that will provide a certain encoding speed to compression ratio. A slower preset will provide better compression (compression is quality per filesize). This means that, for example, if you target a certain file size or constant bit rate, you will achieve better quality with a slower preset. Similarly, for constant quality encoding, you will simply save bitrate by choosing a slower preset.

The general guideline is to use the slowest preset that you have patience for. Current presets in descending order of speed are: `ultrafast`, `superfast`, `veryfast`, `faster`, `fast`, `medium`, `slow`, `slower`, `veryslow`, `placebo`. The default preset is `medium`. Ignore `placebo` as it is not useful (see FAQ). You can see a list of current presets with `-preset help`, and what settings they apply with `x264 --fullhelp`.

You can optionally use `-tune` to change settings based upon the specifics of your input. Current tunings include: `film`, `animation`, `grain`, `stillimage`, `psnr`, `ssim`, `fastdecode`, `zerolatency`. For example, if your input is animation then use the `animation` tuning, or if you want to preserve grain then use the `grain` tuning. If you are unsure of what to use or your input does not match any of tunings then omit the `-tune` option. You can see a list of current tunings with `-tune help`, and what settings they apply with `x264 --fullhelp`.

Another optional setting is `-profile:v` which will limit the output to a specific H.264 profile. This can generally be omitted unless the target device only supports a certain profile (see Compatibility). Current profiles include: `baseline`, `main`, `high`, `high10`, `high422`, `high444`. Note that usage of `-profile:v` is incompatible with lossless encoding.

As a shortcut, you can also list all possible internal presets/tunes for FFmpeg by specifying no preset or tune option at all:

```
ffmpeg -y -i input -c:v libx264 -preset -tune -f mp4 /dev/null
```

> **Note:** Windows users should use `NUL` instead of `/dev/null`.

3. Use your settings

Once you've chosen your settings apply them for the rest of your videos if you are encoding more. This will ensure that they will all have similar quality.

CRF Example

```
ffmpeg -i input -c:v libx264 -preset slow -crf 22 -c:a copy output.mkv
```

Note that in this example the audio stream of the input file is simply stream copied over to the output and not re-encoded.

Two-Pass

This method is generally used if you are targeting a specific output file size and output quality from frame to frame is of less importance. This is best explained with an example. Your video is 10 minutes (600 seconds) long and an output of 50 MB is desired. Since `bitrate = file size / duration`:

```
(50 MB * 8192 [converts MB to kilobits]) / 600 seconds = ~683 kilobits/s total bitrate
683k - 128k (desired audio bitrate) = 555k video bitrate
```

Two-Pass Example

```
ffmpeg -y -i input -c:v libx264 -preset medium -b:v 555k -pass 1 -an -f mp4 /dev/null && \
ffmpeg -i input -c:v libx264 -preset medium -b:v 555k -pass 2 -c:a libfdk_aac -b:a 128k output.mp4
```

> **Note:** Windows users should use `NUL` instead of `/dev/null`.

As with CRF, choose the slowest preset you can tolerate.

Also see Making a high quality MPEG-4 ("DivX") rip of a DVD movie. It is an MEncoder guide, but it will give you an insight about how important it is to use two-pass when you want to efficiently use every bit when you're constrained with storage space.

Lossless H.264

You can use `-qp 0` or `-crf 0` to encode a lossless output. Use of `-qp` is recommended over `-crf` for lossless because 8-bit and 10-bit x264 use different `-crf` values for lossless. Two useful presets for this are `ultrafast` or `veryslow` since either a fast encoding speed or best compression are usually the most important factors. Most non-FFmpeg based players will not be able to decode lossless (but YouTube can), so if compatibility is an issue you should not use lossless.

Lossless Example (fastest encoding)

```
ffmpeg -i input -c:v libx264 -preset ultrafast -qp 0 output.mkv
```

Lossless Example (best compression)

```
ffmpeg -i input -c:v libx264 -preset veryslow -qp 0 output.mkv
```

Overwriting default preset settings

You can overwrite default preset settings with the `x264opts` option, the `x264-params` option, or by using the libx264 private options (see `ffmpeg -h encoder=libx264`). This is not recommended unless you know what you are doing. The presets were created by the x264 developers and tweaking values to get a better output is usually a waste of time.

Example:

```
ffmpeg -i input -c:v libx264 -preset slow -crf 22 -x264opts keyint=123:min-keyint=20 -c:a copy output.m
```

Additional Information & Tips

ABR (Average Bit Rate)

```
ffmpeg -i input -c:v libx264 -b:v 1000k output.mp4
```

This provides something of a "running average" target, with the end goal that the final file match this number "overall on average" (so basically, if it gets a lot of black frames, which cost very little, it will encode them with less than the requested bitrate, but then the next few seconds of (non-black) frames it will encode at very high quality, to bring the average back in line). Using 2-pass can help this method to be more effective. You can also use this in combination with a "max bit rate" setting in order to prevent some of the swings.

CBR (Constant Bit Rate)

There is no native CBR mode, but you can "simulate" a constant bit rate setting by tuning the parameters of ABR:

```
ffmpeg -i input -c:v libx264 -b:v 4000k -minrate 4000k -maxrate 4000k -bufsize 1835k out.m2v
```

In the above example, `-bufsize` is the "rate control buffer" so it will enforce your requested "average" (4000k in this case) across each 1835k worth of video. So basically it is assumed that the receiver/end player will buffer that much data so it's ok to fluctuate within that much.

Of course, if it's all just empty/black frames then it will still serve less than that many bits/s but it will raise the quality level as much as it can, up to the crf level.

CRF with maximum bit rate

You can also also use a crf with a maximum bit rate by specifying both crf *and* maxrate settings, like

```
ffmpeg -i input -c:v libx264 -crf 20 -maxrate 400k -bufsize 1835k output.mp4
```

This will effectively "target" crf 20, but if the output exceeds 400kb/s, it will degrade to something less than crf 20 in that case.

Low Latency

libx264 offers a `-tune zerolatency` option. See the StreamingGuide.

Compatibility

All devices

If you want your videos to have highest compatibility with target devices (older iOS versions or all Android devices):

```
-profile:v baseline -level 3.0
```

This disables some advanced features but provides for better compatibility. Typically you may not need this setting (and therefore avoid using `-profile:v` and `-level`), but if you do use this setting it may increase the bit rate quite a bit compared to what is needed to achieve the same quality in higher profiles.

iOS

iOS Compatability (source)			
Profile	**Level**	**Devices**	**Options**
Baseline	3.0	All devices	`-profile:v baseline -level 3.0`
Baseline	3.1	iPhone 3G and later, iPod touch 2nd generation and later	`-profile:v baseline -level 3.1`
Main	3.1	iPad (all versions), Apple TV 2 and later, iPhone 4 and later	`-profile:v main -level 3.1`
Main	4.0	Apple TV 3 and later, iPad 2 and later, iPhone 4S and later	`-profile:v main -level 4.0`
High	4.0	Apple TV 3 and later, iPad 2 and later, iPhone 4S and later	`-profile:v high -level 4.0`
High	4.1	iPad 2 and later and iPhone 4S and later	`-profile:v high -level 4.1`

This table does not include any additional restrictions which may be present.

Apple Quicktime

Apple QuickTime only supports YUV planar color space with 4:2:0 chroma subsampling (use `-pix_fmt yuv420p`) for H.264 video. For information on additional restrictions see QuickTime-compatible Encoding.

Blu-ray

See Authoring a professional Blu-ray Disc with x264.

Pre-testing your settings

Encode a random section instead of the whole video with the `-ss` and `-t` options to quickly get a general idea of what the output will look like.

- `-ss`: Offset time from beginning. Value can be in seconds or HH:MM:SS format.
- `-t`: Output duration. Value can be in seconds or HH:MM:SS format.

`faststart` for web video

You can add `-movflags +faststart` as an output option if your videos are going to be viewed in a browser. This will move some information to the beginning of your file and allow the video to begin playing before it is completely downloaded by the viewer. It is not required if you are going to use a video service such as YouTube.

FAQ

Will two-pass provide a better quality than CRF?

No, though it does allow you to target a file size more accurately.

Why is `placebo` a waste of time?

It helps at most ~1% compared to the `veryslow` preset at the cost of a much higher encoding time. It's diminishing returns: `veryslow` helps about 3% compared to the `slower` preset, `slower` helps about 5% compared to the `slow` preset, and `slow` helps about 5-10% compared to the `medium` preset.

Why doesn't my lossless output look lossless?

Blame the RGB to YUV color space conversion. If you convert to yuv444 it should still be lossless (which is the default now).

Will a graphics card make x264 encode faster?

Not necessarily. x264 supports OpenCL for some lookahead operations. There are also some proprietary encoders that utilize the GPU, but that does not mean they are well optimized, though encoding time may be faster; and they might be worse than vanilla x264, and possibly slower. Regardless, FFmpeg today doesn't support any means of GPU encoding, outside of libx264.

Encoding for dumb players

You may need to use `-pix_fmt yuv420p` for your output to work in QuickTime and most other players. These players only supports the YUV planar color space with 4:2:0 chroma subsampling for H.264 video. Otherwise, depending on your source, ffmpeg may output to a pixel format that may be incompatible with these players.

Additional Resources

- x264 Settings - MeWiki
- x264 Encoding Suggestions - MeWiki
- Constant Rate Factor Guide (copy of a deleted Handbrake Wiki page)

Last modified 3 weeks ago

Download in other formats:
Plain Text

wiki: Scaling (resizing) with ffmpeg

FFmpeg has got a very powerful scale filter, which can be used to accomplish various tasks. Some of them are listed here. In all the examples, the starting image (input.jpg) will be this one (535x346 pixels):

If you need to simply resize your video to a specific size (e.g 320x240), you can use the scale filter in its most basic form:

```
ffmpeg -i input.avi -vf scale=320:240 output.avi
```

Same works for images too:

```
ffmpeg -i input.jpg -vf scale=320:240 output_320x240.png
```

The resulting image will look like this:

As you can see, the aspect ratio is not the same as in the original image, so the image appears stretched. If we'd like to keep the aspect ratio, we need to specify only one component, either width or height, and set the other component to -1. For example, this command line:

```
ffmpeg -i input.jpg -vf scale=320:-1 output_320.png
```

will set the width of the output image to 320 pixels and will calculate the height of the output image according to the aspect ratio of the input image. The resulting image will have a dimension of 320x207 pixels.

There are also some useful constants which can be used instead of numbers, to specify width and height of the output image. For example, if you want to stretch the image in such a way to only double the width of the input image, you can use something like this (iw = input width constant, ih = input height constant):

```
ffmpeg -i input.jpg -vf scale=iw*2:ih input_double_width.png
```

The output image will look like this:

Sometimes there is a need to scale the input image in such way it fits into a specified rectangle, i.e. if you have a placeholder (empty rectangle) in which you want to scale any given image. This is a little bit tricky, since you need to check the original aspect ratio, in order to decide which component to specify and to set the other component to -1 (to keep the aspect ratio). For example, if we would like to scale our input image into a rectangle with dimensions of 320x240, we could use something like this:

```
ffmpeg -i input.jpg -vf scale="'if(gt(a,4/3),320,-1)':'if(gt(a,4/3),-1,240)'" output_320x240_boxed.png
```

The output image will look like this:

The area below the image is shaded with boxes to show there was some additional space left in the box, due to the original image not having the same aspect ratio as the box, in which the image was supposed to fit.

Of course, this approach is only used when your input image is not known in advance, because if it was known, you would easily figure out if you need to use either:

```
-vf scale=320,-1
```

or

```
-vf scale=-1,240
```

Last modified 7 months ago

Attachments (5)

- input.jpg (53.8 KB) - added by *burek* 9 months ago.
- output_320x240.png (169.5 KB) - added by *burek* 9 months ago.
- output_320.png (147.9 KB) - added by *burek* 9 months ago.
- input_double_width.png (577.6 KB) - added by *burek* 9 months ago.
- output_320x240_boxed.png (167.5 KB) - added by *burek* 9 months ago.

Download all attachments as: .zip

Download in other formats:
Plain Text

Using ffmpeg to manipulate audio and video files

How to tame the "Swiss army knife" of audio and video manipulation...

FOREWORD

Since I put up this page I've had loads of e-mails asking how to do this, that and the other with ffmpeg. As is mentioned later on this page, I'm no ffmpeg expert so I'm not really the person to ask and *such questions will be ignored*. Please visit this page.

INTRODUCTION

I recently became curious about converting between video formats, wondering if there was anything I could do to reduce the size of the various .mpg, .flv, .avi and .mov files that I have lying around. I also wondered if it would be possible to convert some of these files to the .3gp format used by 3G cellphones, and if it would be possible to rip a DVD-Video to DivX. Finally, I wanted to know if I could rip DVDs of concerts and operas that I have and listen to them on my MP3 player. Best of all would be the ability to do all of this with a single tool.

Well, it is possible, and that single tool is ffmpeg.

This tutorial should enable you to install ffmpeg and the auxiliary libraries that will give ffmpeg support for various codecs. It'll then go on to explain the basics of what a video file is, how it's created by ffmpeg and how a media player takes it apart again to display the picture and give you the sound. Next, you'll find out how to influence the way the data is produced. Finally, I'll deal with more advanced topics such as merging several sources and introducing time differentials.

What you will *not* find out in this tutorial is the usage of absolutely every option available to ffmpeg, nor will you find out details of any particular codecs or container formats. Complete mastery of the subject is beyond the scope of this howto and certainly well beyond my own knowledge. If you want to learn more then I'm sure there's plenty of reading available on the 'Net – not to mention ffmpeg's source code itself.

DEPENDENCIES

Support for many codecs and file formats is already provided by libavformat and libavcodec, two libraries that are supplied in ffmpeg's source. However, if you want support for a few more common (and not so common) codecs and formats then you'll need to install a few external libraries and get ffmpeg to use them. Refer to the sites linked to here and/or to instructions contained in the source tarballs for details on how to install the following pieces of software:

- MP3
  Support for MP3 audio is provided by the LAME MPEG Audio Layer-III encoder library, which is available here: http://lame.sourceforge.net.
- GSM
  If, like me, you use a system that e-mails you GSM audio attachments of messages left in your voicemail then you'll need libgsm to convert them and play them back. libgsm is in fact a bit of a pig to track down so I'm providing it here:
  libgsm-1.0.13.tar.bz2
  libgsm-1.0.13.tar.bz2.md5
- DivX
  DivX is a variant of MPEG4. While ffmpeg can produce MPEG4 on its own, it requires an external library to produce something to the specifications of DivX, and it's the XviD library that takes care of this. XviD is available from xvid.org.
- AMR
  3G cellphones use the AMR wideband and narrowband codecs for audio streams in .3gp video files. These codecs are available from the 3gpp.org website (if you have the patience to find them) but their installation is not straightforward. A kind soul in the Czech Republic, Stanislav Brabec, has bundled the codecs into Linux-friendly libraries that are used by ffmpeg. See here.
- OGG/Vorbis
  Vorbis is a recently-developed audio codec similar to MP3 in its working but totally free, unlike MP3. There are licensing issues around MP3 which mean that the owner of the technology, the Fraunhofer Institute, began imposing a fee of $15,000.00 plus $5.00 per encoder and 50 cents per player sold or distributed. [Ref. Wired magazine Sept. 2001, pg 74 article by Pete Rojas]. The Vorbis audio codec is managed by the libvorbis library available from xiph.org. The encoded data is packaged into .ogg files by libogg, also available from the same page.

- AAC – Advanced Audio Coding
  Also known as MPEG-4 audio, AAC is used by many mainstream media devices such as the iPod and most 3G cellphones. In order to convert to/from formats using AAC, you'll need two libraries. libfaad takes care of *de*coding AAC while libfaac takes care of *en*coding it. These libraries are available from sourceforge.net.

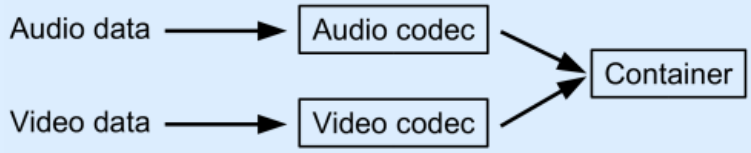
- Subversion
  ffmpeg's source code is distributed from a Subversion repository. You'll need the Subversion client (svn) to grab it. See subversion.tigris.org.

THE BASICS OF AUDIO/VIDEO

In reality, the subject is far more complex than I'm about to describe here. This said, you won't really need to know more than these basics for everyday use of ffmpeg.

The first hurdle you need to get over is the fact that the type of file you're dealing with (.avi, .mpg, .mov etc.) isn't the only factor to take into account. I wouldn't need to work now if I was given a penny each time someone said to me something like *"I have two AVI files. One plays back and the other doesn't, and this stupid machine complains about missing code-c's or something like that. What do I do?"*

This diagram should help explain what's going on:



Audio and video would use up huge amounts of storage space if they were stored raw. Assuming an NTSC standard video at 720x480 pixels, 30 frames per second and 24-bit RGB color, we're talking about 1,036,800 bytes per frame. That's nearly 30MB per second, or over 200GB for a 2-hour movie. And that's just the video… Something needs to be done so that the movie can be stored on a consumer-grade medium such as a Digital Versatile Disc (DVD). The data needs to be compressed.

Traditional, lossless compression algorithms such as ZIP, gz and bzip2 don't get anywhere near reducing the size of the data enough, so we need to look into lossy compression, ie: compression that is far more efficient but with a trade-off in that the picture quality suffers. The same applies to audio with a degradation of the sound quality. The object of much research over the past few decades has been the elaboration of new algorithms that manage to reduce the quantity of data needed for the audio and video while also reducing the damage done to the picture and sound as a result of the compression. These algorithms that allow us to encode the data in order to transport it, and to decode the data the other end, are called **codecs**. Several codecs are contained in the libavcodec library supplied with ffmpeg – others are provided by LAME, libgsm, XviD, AMR and libvorbis.

Once the audio and video streams have been encoded by their respective codecs, this encoded data needs to be encapsulated into a single file. This file is called the **container**. One particular type of container is AVI (Audio-Video Interleaved). AVI is only a method of mixing the encoded audio and video together in a single file. It contains data informing the media player trying to play the file back what audio and video codecs were used, and there are many different possible combinations of codecs that can be used within each type of container, which explains why a system may play back some AVI files and not others. The system can extract the encoded audio and video data from the AVI file no problem, but can it decode that encoded data once it has done so? Only if the required codecs are present.

The algorithms used to pack the encoded audio and video data into various types of containers are mostly contained in the libavformat library supplied with ffmpeg. Support for OGG files containing Vorbis-encoded audio data is supplied by libogg.

Playback of a multimedia file operates in the exact opposite direction. All you have to do is refer to the diagram above having reversed the direction of all the arrows. Once the container used is identified, it tells us which codecs are needed to decode the data. The audio and video streams are then extracted from the container and fed through the appropriate codecs, and out the other end we get raw audio and video data that can be fed to the audio and display subsystems of the computer.

WHAT IS FFMPEG, THEN?

ffmpeg is a tool that, in its simplest form, implements a decoder and then an encoder, thus enabling the user to convert files from one container/codec combo to another, for example a VOB file from a DVD containing MPEG2 video and AC3 audio to an AVI file containing MPEG4 video and MP3 audio, or a QuickTime file containing SVQ3 video and MP3 audio to a 3GP file containing H263 video and AMR wideband audio. The original container is examined and the encoded data extracted and fed through the codecs. The newly-decoded data is then fed through the "target" codecs into the new container.

ffmpeg can also do a few basic manipulations on the audio and video data just before it is re-encoded by the target codecs. These manipulations include changing the sample rate of the audio and advancing or delaying it with respect to the video. They also include changing the frame rate of the resulting video, cropping it, resizing it, placing bars left and right and/or top and bottom in order to pad it when necessary, or changing the aspect ratio of the picture (if the container supports anamorphic display – not all do). Furthermore, ffmpeg allows importing audio and video from different sources, thus allowing dubbing for example.

Another piece of software similar to ffmpeg is transcode, but it seems far less flexible to me than ffmpeg when it comes to manipulating audio and video streams. Parts of transcode, though, are invaluable for grabbing data off DVD-Videos, in particular tcprobe (used for getting information about titles on the DVD), tccat (for descrambling – if libdvdcss is installed – and extracting the VOB files from the DVD, you'll also need libdvdread) and sometimes tcextract for extracting the various audio, video and subtitle streams from the VOB files. This said, mplayer (built on ffmpeg) is perfectly capable of extracting titles from a DVD-Video and ffmpeg can extract the individual channels from the resulting .vob file.

BUILDING FFMPEG

First of all you need to grab the source code. Go to http://ffmpeg.mplayerhq.hu/download.html and follow the instructions for getting the latest SVN release.

Once that's done we need to configure the build process using an autoconf-like (but not actually based on autoconf in the case of ffmpeg) script. Don't forget to add a --prefix option if you're planning on installing ffmpeg in an area other than /usr/local, and --extra-cflags and --extra-libs options if you've installed any of the dependencies in areas other than /usr and /usr/local.

cd into the ffmpeg directory and run the following:

```
$ ./configure --enable-gpl --enable-liba52 --enable-libgsm --enable-libxvid \
--enable-libamr_nb --enable-libamr_wb --enable-libmp3lame --enable-libogg \
--enable-libvorbis --enable-libfaac --enable-libfaad --enable-shared
```

Explanations:

- --enable-gpl
  ffmpeg is usually released under the GNU Lesser General Public Licence (LGPL). However, some modules that *can* be included in it, such as liba52 for example, are released under the GNU General Public Licence (GPL). If these optional modules are included then the GPL must apply to the whole of ffmpeg and the libavcodec and libavformat libraries. This option enables the building of a GPL'ed version of ffmpeg et al.
- --enable-liba52
  This (optional because it modifies the licence of ffmpeg) library, liba52, which is part of libavcodec, contains the codec used to decode and encode AC3 audio commonly used on DVD-Videos.
- --enable-libgsm --enable-libxvid
  Tells ffmpeg to use these two dependencies installed earlier.
- --enable-libamr_nb --enable-libamr_wb
  Tells ffmpeg to include support for AMR narrowband and wideband.
- --enable-libmp3lame --enable-libogg --enable-libvorbis --enable-libfaac --enable-libfaad
  Tells ffmpeg to use these dependencies installed earlier.
- --enable-shared
  This tells the compiler to build libavformat, libavcodec and libavutils as shared libraries. If you're planning on simply using ffmpeg in order to transcode audio and video files then it doesn't really matter whether you build ffmpeg this way or not. On the other hand, if you're planning on writing your own multimedia software that incorporates these libraries then you'll be making things easier for yourself in the long run by building them shared and linking them dynamically into your executable.

You should end up with output similar to this:

```
install prefix            /usr/local
source path               /home/howard/src/ffmpeg
C compiler                gcc
make                      make
.align is power-of-two    no
ARCH                      x86_32 (generic)
big-endian                no
MMX enabled               yes
CMOV enabled              no
CMOV is fast              no
gprof enabled             no
debug symbols             yes
strip symbols             yes
optimize                  yes
static                    yes
shared                    no
postprocessing support    no
software scaler enabled   no
video hooking             yes
Imlib2 support            no
FreeType support          yes
network support           yes
IPv6 support              yes
threading support         no
SDL support               yes
Sun medialib support      no
AVISynth enabled          no
liba52 support            yes
liba52 dlopened           no
libamr-nb support         yes
libamr-wb support         yes
libfaac enabled           yes
libfaad enabled           yes
libfaad dlopened          no
libgsm enabled            yes
libmp3lame enabled        yes
libnut enabled            no
libogg enabled            yes
libtheora enabled         no
libvorbis enabled         yes
x264 enabled              no
XviD enabled              yes
zlib enabled              yes
License: GPL
Creating config.mak and config.h...
```

This recaps the options with which ffmpeg will be built. We're now ready to build ffmpeg:

```
$ make
```

Depending on the speed of your machine it can take a while for ffmpeg to build. On mine (Celeron 2.93GHz with 1GB RAM) it takes about 5 minutes. Once it's finished building and if there are no errors, we can install it. If there are errors you'll need to ensure that the various libraries

that you asked ffmpeg to use are available in standard directories (/usr and /usr/local). If they're elsewhere you'll have to go back to the ./configure command and add the appropriate --extra-cflags (to add the correct include path) and --extra-libs (to tell the linker where to look for the libraries) options. If that still doesn't work you can always join the ffmpeg users' mailing list and ask for help.

Next you want to remove any traces of previous versions of the shared libraries built just now:

```
$ su -c "rm -f /usr/local/lib/libav{format,codec,util}.*"
Password: (enter your root password here)
  ... or for some distros:
$ sudo rm -f /usr/local/lib/libav{format,codec,util}.*
```

If you selected an installation directory that doesn't require extra privileges for you to write in it:

```
$ make install
```

If you need root access to write to the installation path:

```
$ su -c "make install"
Password: (enter your root password here)
  ... or for some distros:
$ sudo make install
```

ffmpeg is now installed and you can start using it.

BASIC AUDIO TRANSCODING

Start by ripping an audio track from a music CD and calling the resulting file audio.wav. There are plenty of documents on the 'Net explaining how to do this if you don't yet know how to.

Next, get ffmpeg to identify the file. The easiest way is to tell ffmpeg to use it for its input and not tell it to do anything with it. The duration you'll see in the output below will obviously be different from my example unless by some fluke you're using exactly the same track of the same CD (track 15, The Forgotten pt. II, of Joe Satriani's "Flying In A Blue Dream" for those who are curious).

```
$ ffmpeg -i audio.wav
FFmpeg version SVN-r9607, Copyright (c) 2000-2007 Fabrice Bellard, et al.
  configuration:  {snipped for brevity}
  libavutil version: 49.4.1
  libavcodec version: 51.40.4
  libavformat version: 51.12.1
  built on Jul 12 2007 20:22:46, gcc: 3.4.6
Input #0, wav, from 'audio.wav':
  Duration: 00:05:08.1, start: 0.000000, bitrate: 1411 kb/s
  Stream #0.0: Audio: pcm_s16le, 44100 Hz, stereo, 1411 kb/s
Must supply at least one output file
```

Okay then. the track is 5 mins 8.1 seconds long, the "codec" used (in quotes because the sound isn't compressed at all in the case of audio files ripped from a CDDA) is PCM (Pulse Code Modulated) with signed, 16-bit, little-endian samples. The sample rate is 44100Hz, there are two audio channels (stereo) and the resulting bitrate is 1411kbit/s.

Fair enough. Let's convert this to an MP3 file:

```
$ ffmpeg -i audio.wav -acodec mp3 -ab 192k audio.mp3
FFmpeg version SVN-r9607, Copyright (c) 2000-2007 Fabrice Bellard, et al.
  configuration:  {snipped for brevity}
  libavutil version: 49.4.1
  libavcodec version: 51.40.4
  libavformat version: 51.12.1
  built on Jul 12 2007 20:22:46, gcc: 3.4.6
Input #0, wav, from 'audio.wav':
  Duration: 00:05:08.1, start: 0.000000, bitrate: 1411 kb/s
  Stream #0.0: Audio: pcm_s16le, 44100 Hz, stereo, 1411 kb/s
Output #0, mp3, to 'audio.mp3':
  Stream #0.0: Audio: mp3, 44100 Hz, stereo, 192 kb/s
Stream mapping:
  Stream #0.0 -> #0.0
Press [q] to stop encoding
[mp3 @ 0x852f018]lame: output buffer too small (buffer index: 592, free bytes: 1712)
size=    7221kB time=308.1 bitrate= 192.0kbits/s
video:0kB audio:7221kB global headers:0kB muxing overhead 0.000000%
```

The error message about the output buffer being too small is nothing important. It's merely a result of the way data is passed between ffmpeg and the external codec, LAME. This is what you should have right now:

```
$ ls -ls
total 60384
 7236 -rw-r--r--  1 howard users  7394742 2006-10-10 23:36 audio.mp3
53148 -rw-r--r--  1 howard users 54363436 2006-10-10 23:20 audio.wav
```

See how small the MP3 file is compared to the original, uncompressed WAV file.

Explanations:

- -i audio.wav

This tells ffmpeg that we want it to take audio.wav and process it.

- -acodec mp3
  This tells ffmpeg to use the "mp3" audio codec to create the target file.
- -ab 192k
  This tells ffmpeg to use an audio bitrate of 192 kbit/s. The higher this value, the better the audio quality, but the larger the resulting file. 192 kbit/s is pretty good quality audio.
- audio.mp3
  Dump the encoded audio data into a file called audio.mp3.

Personally I prefer to compress audio to OGG/Vorbis rather than MP3. For one thing the codec is totally unencumbered with legal crap, and for another it gives far better results than MP3 at equal bitrates. This is how I'd convert audio.wav to OGG/Vorbis using ffmpeg:

```
$ ffmpeg -i audio.wav -acodec vorbis -aq 60 audio.ogg
```

Explanations:

- Vorbis is a variable bit rate (VBR) codec, which means that you tell it to create audio of a given quality and it uses however many bytes of data it takes to achieve that goal. The (undocumented in the man page!) "-aq" option of ffmpeg is how you tell it what audio quality to ask of a VBR codec, and the numerical value you give is interpreted differently by different codecs. In the case of the Vorbis codec, "0" is the poorest quality available, getting better with larger values. The value of "60" used here results in audio at roughly 160 kbit/s, which is very good.

So far so good. Now let's resample the audio at 22050 Hz instead of 44100 Hz, convert it to mono, use a much lower bitrate and generate an MP2 file instead of MP3:

```
$ ffmpeg -i audio.wav -acodec mp2 -ac 1 -ar 22050 -ab 64k audio.mp2
FFmpeg version SVN-r9607, Copyright (c) 2000-2007 Fabrice Bellard, et al.
  configuration:  {snipped for brevity}
  libavutil version: 49.4.1
  libavcodec version: 51.40.4
  libavformat version: 51.12.1
  built on Jul 12 2007 20:22:46, gcc: 3.4.6
Input #0, wav, from 'audio.wav':
  Duration: 00:05:08.1, start: 0.000000, bitrate: 1411 kb/s
  Stream #0.0: Audio: pcm_s16le, 44100 Hz, stereo, 1411 kb/s
Output #0, mp2, to 'audio.mp2':
  Stream #0.0: Audio: mp2, 22050 Hz, mono, 64 kb/s
Stream mapping:
  Stream #0.0 -> #0.0
Press [q] to stop encoding
size=    2407kB time=308.1 bitrate=  64.0kbits/s
video:0kB audio:2407kB global headers:0kB muxing overhead 0.000000%
```

Notice how much smaller audio.mp2 is than audio.mp3.

Explanations:

- -ac 1
  Tells ffmpeg to downmix the audio input to 1 audio channel (mono) instead of leaving it as the input (stereo, 2 channels).
- -ar 22050
  Tells ffmpeg to resample the audio at 22050 Hz instead of leaving it at 44100 Hz.

Now let's convert this MP2 file back to WAV PCM but with unsigned 8-bit samples instead of 16-bit. Reminder, WAV PCM is uncompressed so we don't need to specify a bit rate:

```
$ ffmpeg -i audio.mp2 -acodec pcm_u8 audio.8bit.wav
FFmpeg version SVN-r9607, Copyright (c) 2000-2007 Fabrice Bellard, et al.
  configuration:  {snipped for brevity}
  libavutil version: 49.4.1
  libavcodec version: 51.40.4
  libavformat version: 51.12.1
  built on Jul 12 2007 20:22:46, gcc: 3.4.6
Input #0, mp3, from 'audio.mp2':
  Duration: 00:05:08.1, start: 0.000000, bitrate: 63 kb/s
  Stream #0.0: Audio: mp2, 22050 Hz, mono, 64 kb/s
Output #0, wav, to 'audio.8bit.wav':
  Stream #0.0: Audio: pcm_u8, 22050 Hz, mono, 176 kb/s
Stream mapping:
  Stream #0.0 -> #0.0
Press [q] to stop encoding
size=    6635kB time=308.1 bitrate= 176.4kbits/s
video:0kB audio:6635kB global headers:0kB muxing overhead 0.000648%
```

Note that running the following command will give you a list of the codecs and formats that ffmpeg can work with:

```
$ ffmpeg -formats
```

What this section should have taught you:

- The "-i" option tells ffmpeg what file to use for input. This is the file that you want to convert into something else.

- You can tell ffmpeg which audio codec to use in the target file with the "-acodec" option. The special case "-acodec copy" tells ffmpeg to use the same codec to encode as was used to decode. In other words, no transcoding of the audio occurs.
- It is possible to resample the audio on the fly using the "-ar" option, to set the bit rate and thus the quality of the resulting file with the "-ab" option, and to downmix stereo to mono (or even Dolby 5:1 to stereo or mono) with the "-ac" option.

BASIC VIDEO TRANSCODING

Let's start by grabbing a Flash Video .flv file from YouTube. This can't be done directly so we need to enlist the help of a third party site to do this. Go to http://javimoya.com/blog/youtube_en.php and enter this URL into the input zone at the top of the page:

http://www.youtube.com/watch?v=orLpZxEd24Y

Your browser will then download a file that you should rename to kitty.flv since this is a video showing the musical purrrrrrowess of a young, furry, feline composer.

Let's see what ffmpeg can tell us about this video:

```
$ ffmpeg -i kitty.flv
{output snipped}
Seems that stream 1 comes from film source: 1000.00 (1000/1) -> 25.00 (25/1)
Input #0, flv, from 'kitty.flv':
  Duration: 00:01:43.4, start: 0.000000, bitrate: N/A
  Stream #0.0: Audio: mp3, 22050 Hz, mono
  Stream #0.1: Video: flv, yuv420p, 320x240, 25.00 fps(r)
```

This shows that the audio stream is a mono track sampled at 22050 Hz and MP3-encoded. The video stream was encoded using the "flv" (Flash Video) codec, "yuv420p" is how the color is encoded, the picture resolution is 320x240 pixels and the frame rate is 25 frames per second. Kitty is about to make his TV début so we need to have a copy of this file in a form that can be used for broadcast. An NTSC MPEG file that would otherwise end up on a DVD will do fine:

```
$ ffmpeg -i kitty.flv -target ntsc-dvd -aspect 4:3 kitty.mpg
{snipped}
Seems that stream 1 comes from film source: 1000.00 (1000/1) -> 25.00 (25/1)
Input #0, flv, from 'kitty.flv':
  Duration: 00:01:43.4, start: 0.000000, bitrate: N/A
  Stream #0.0: Audio: mp3, 22050 Hz, mono
  Stream #0.1: Video: flv, yuv420p, 320x240, 25.00 fps(r)
Output #0, dvd, to 'kitty.mpg':
  Stream #0.0: Video: mpeg2video, yuv420p, 720x480, q=2-31, 6000 kb/s, 29.97 fps(c)
  Stream #0.1: Audio: ac3, 48000 Hz, mono, 448 kb/s
Stream mapping:
  Stream #0.1 -> #0.0
  Stream #0.0 -> #0.1
Press [q] to stop encoding
frame= 3095 q=2.0 Lsize=   45130kB time=103.2 bitrate=3581.1kbits/s
video:31666kB audio:5660kB global headers:0kB muxing overhead 20.909333%
```

Explanations:

- -target ntsc-dvd
  "-target" is one of the most useful options of ffmpeg. It instructs ffmpeg to just "do what it takes" for the target file to be usable on an NTSC DVD-Video in this case (there are other pre-defined targets). This includes selecting the correct audio and video codecs, appropriate bit rates, image size, frame rate and other parameters. Indeed, if we look at the lines in the "Output #0" section of the output above, we see that the video is set to MPEG2 video, yuv420p color encoding, 720x480 pixel image size, 6 mbit/s bit rate, 29.97 frames per second (the q=2-31 part is the admissible quality levels, 2, the best, to 31). The audio stream is ac3-encoded with a sampling rate of 48 KHz, mono at 448 kbit/s. These are standard parameters for a DVD-Video with a mono soundtrack and can be overridden.
- -aspect 4:3
  MPEG2 video allows for anamorphic display. This means that the aspect ratio (width divided by the height) of the physical picture displayed on-screen can be different from the ratio obtained by dividing the pixel-width of the image by its pixel-height. This has to be built in to the video specification used with DVDs because, regardless of whether we're viewing a movie in old 4:3 aspect ratio or a movie in 16:9 widescreen format, the pixel size of the digital image is still 720x480 for an NTSC DVD (720x576 for PAL). The image is in fact stretched such that its aspect ratio is as specified by the "-aspect" option. This usually involves stretching it only along its width or along its height, hence the name "anamorphic".

Given that the original was a movie at a resolution of 320x240, we really don't need the full 6 mbit/s bit rate, especially as the codec wasn't even able to use the full bandwidth and ended up limiting itself to about 3.5 mbit/s. We can try and create a smaller file by getting the codec to try and keep to 2 mbit/s. The DVD standard also allows for MP2-encoded audio, so let's override both the video bitrate and the audio codec used:

```
$ ffmpeg -i kitty.flv -target ntsc-dvd -b 2000k -acodec mp2 -ar 22050 -ab 64k -aspect 4:3 kitty2.mpg
{snipped}
Seems that stream 1 comes from film source: 1000.00 (1000/1) -> 25.00 (25/1)
Input #0, flv, from 'kitty.flv':
  Duration: 00:01:43.4, start: 0.000000, bitrate: N/A
  Stream #0.0: Audio: mp3, 22050 Hz, mono
  Stream #0.1: Video: flv, yuv420p, 320x240, 25.00 fps(r)
Output #0, dvd, to 'kitty2.mpg':
  Stream #0.0: Video: mpeg2video, yuv420p, 720x480, q=2-31, 2000 kb/s, 29.97 fps(c)
  Stream #0.1: Audio: mp2, 22050 Hz, mono, 64 kb/s
Stream mapping:
  Stream #0.1 -> #0.0
  Stream #0.0 -> #0.1
Press [q] to stop encoding
frame= 3095 q=4.7 Lsize=   27174kB time=103.2 bitrate=2156.3kbits/s
```

```
video:21521kB audio:809kB global headers:0kB muxing overhead 21.696595%
```

This is what we have so far:

```
27208 -rw-r--r--  1 howard users 27826176 2006-10-12 21:36 kitty2.mpg
 4252 -rw-r--r--  1 howard users  4340341 2006-10-11 13:31 kitty.flv
45184 -rw-r--r--  1 howard users 46213120 2006-10-12 13:25 kitty.mpg
```

The second MPEG file is much smaller than the first one, but they're both much bigger than the original Flash Video file. That is because MPEG2 is nowhere near as efficient a codec as some others, not to mention the increased image size (720x480 over 320x240). What we can do, though, is to try a different video codec, like MPEG4 for example. Now, for reasons that I don't understand – after all I'm no expert in this area – mplayer is unable to play back an MPEG4 video created directly from a Flash Video, while ffplay (which is part of ffmpeg) can. And yet, mplayer and ffmpeg share the same codebase. Go figure. So, what we're going to work with from here on in is the first MPEG file, kitty.mpg. Let's transcode it to MPEG4, bring the picture size back down to 320x240 and the frame rate down to 10 fps, see if we can drop the video bit rate to 300 kbit/s, and convert the AC3-encoded, 48000 Hz, 448 kbit/s audio to MP3, 22050 Hz, 64 kbit/s:

```
$ ffmpeg -i kitty.mpg -vcodec mpeg4 -s 320x240 -b 300k -r 10 -acodec mp3 -ar 22050 -ab 64k -f avi kitty.avi
{snipped}
Input #0, mpeg, from 'kitty.mpg':
  Duration: 00:01:42.8, start: 0.500000, bitrate: 3595 kb/s
  Stream #0.0[0x1e0]: Video: mpeg2video, yuv420p, 720x480, 9000 kb/s, 29.97 fps(r)
  Stream #0.1[0x80]: Audio: ac3, 48000 Hz, mono, 448 kb/s
Output #0, avi, to 'kitty.avi':
  Stream #0.0: Video: mpeg4, yuv420p, 320x240, q=2-31, 300 kb/s, 10.00 fps(c)
  Stream #0.1: Audio: mp3, 22050 Hz, mono, 64 kb/s
Stream mapping:
  Stream #0.0 -> #0.0
  Stream #0.1 -> #0.1
Press [q] to stop encoding
frame= 1034 q=4.3 Lsize=    4837kB time=103.4 bitrate= 383.2kbits/s
```

What this section should have taught you:

- The "-target" option tells ffmpeg to set a whole bunch of parameters automatically in order to comply with standards related to the specified target. The list of predefined automatic targets is available in the man page for ffmpeg.
- The parameters defined automatically with a "-target" option can be overridden manually or can all be set manually if the "-target" option isn't used at all.
- The video codec is selected with the "-vcodec" option, the picture size with the "-s" option, the video bitrate with "-b" and the frame rate with "-r".
- The "-f" option selects the container format. "ffmpeg -formats" lists the available formats.

CROPPING AND PADDING THE IMAGE

Let's take our original kitty.flv movie and turn it into a widescreen MPEG movie that can be used on a DVD-Video. The trouble is, its aspect ratio is 4:3, which is narrower than the 16:9 of widescreen format. Its current size is 320x240. If we want to give it a widescreen aspect ratio, its height should be 320/(16/9)=180 pixels instead of 240. If we simply resize the image in order to squash it vertically, the picture will be deformed. The only thing we can do in order to stop the picture from going anamorphic is to trim off parts of it, or to "crop" it. The picture is 60 pixels too "tall" so let's shave 30 pixels off the top and off the bottom, then convert the result to a 16:9 NTSC DVD:

```
$ ffmpeg -i kitty.flv -croptop 30 -cropbottom 30 -target ntsc-dvd -aspect 16:9 kitty.169.mpg
```

Explanations:

- The "-croptop" and "-cropbottom" options slice the given number of pixels off the top and the bottom of the image respectively. There are also "-cropleft" and "-cropright" options for narrowing an image with an aspect ratio that's too high. The number of pixels to slice off **must be an even number**.
- The various "-crop" options act **BEFORE** any resizing takes place. They crop the specified number of pixels off the **ORIGINAL** image.

Now let's turn this 16:9 NTSC video into a 480x360 letterboxed AVI at 15 frames per second. "Letterboxed" means that we keep the whole widescreen image and put bars on the top and bottom so that the resulting bar-image-bar sandwich has the required aspect ratio:

```
$ fmpeg -i kitty.169.mpg -acodec mp3 -ar 44100 -ab 128k -vcodec msmpeg4v2 -b 500k \
-s 480x270 -r 15 -padtop 44 -padbottom 46 -padcolor 000000 -f avi kitty.letterbox.avi
```

Explanations:

- If we want to maintain the same aspect ratio for the image itself and if we want to make it 480 pixels wide, then the height will have to be 480/(16/9)=270 pixels, hence the "-s 480x270".
- Our image being 270 pixels high while we need it to be 360 pixels high in order to obtain the standard 4:3 aspect ratio, we need to slap an extra 90 pixels on it. However, the number of pixels specified in the "-padtop" and "-padbottom" options **has to be an even number**, hence 44 for one and 46 for the other rather than 45 each.
- The color of the padding is specified using the "-padcolor" option. The argument given is a color expressed as three hex bytes just like in HTML (but without the leading "#" sign). See here for a conversion tool that can convert a color into hex notation. There are also "-padleft" and "-padright" options for widening an image with an aspect ratio that's too low for the target.
- The various "-pad" options are applied **AFTER** resizing has taken place. They add the specified number of pixels to the **RESIZED** image.

PROCESSING PORTIONS OF MOVIES

Sometimes you might want to process only the first few seconds or minutes of a movie, either because you're playing with different conversion

parameters and you want to see the results without having to go through with the conversion of the whole DVD or because you're not interested in the rest anyway. The "-t" option of ffmpeg is used to specify how much footage you want to convert. You can either specify the duration as a whole number of seconds (eg: "-t 90" if you want 1½ minutes of video) or you can use hh:mm:ss.sss notation (eg: "-t 00:02:37.200" for 2 minutes and 37.2 seconds of footage).

You don't have to start the conversion at the beginning of the source video either. You can instruct ffmpeg to read a certain time into the movie before it starts converting. This is done with the "-ss" option, and it takes an argument in the same format as "-t" (whole number of seconds or hh:mm:ss.sss). Example: to start converting 3 minutes into the movie and convert 12 minutes of video from then onwards, you'd use "-ss 180" or "-ss 00:03:00", and "-t 720" or "-t 00:12:00".

STRIPPING AUDIO OR VIDEO

It can be desirable to strip the sound track from a movie in order to dub another onto the video, or you might want to do the opposite and extract the audio from a movie (and discard the video) so that you can listen to it on your MP3 player.

Given that you don't actually need to extract the video stream from your movie file in order to dub different audio onto it (we'll see how to do it in the section entitled Multiple Sources), we'll only bother with the second case mentioned. Suffice it to say that the "-an" option tells ffmpeg not to include any audio in the output file, leaving you with a container that includes only video data. Since you'd deal with it technically in exactly the same way as the original movie, the operation of stripping the audio was pretty much a waste of time and disk space.

In order to strip the video stream from the movie file and retain only the audio, you use the "-vn" option. ffmpeg will then output nothing but audio data encoded according to the "-acodec", "-ar", "-ab" and "-ac" options given. As an example of this, let's assume I'm a Jonathan Coulton fan (I do quite like some of his stuff) and I want to grab the audio track off one of the "Mandelbrot Set" movie clips that are up on youtube.com. Fetch the movie in the same way as you did for the kitty composer, only the URL for this one is:

http://www.youtube.com/watch?v=gEw8xpb1aRA

Save the file as "mandelbrot.flv".

The audio quality leaves a bit to be desired since it was compressed rather ruthlessly in order not to bloat the movie file too much, but let's assume I want to burn this song to a CD. This means that I'm going to have to make sure that the audio comes out uncompressed at a sample rate of 44100 Hz in 16-bit samples and 2 channels (stereo). The easiest by far is to conform to the WAV specification since cdrecord can handle these files directly:

```
$ ffmpeg -i mandelbrot.flv -vn -acodec pcm_s16le -ar 44100 -ac 2 mandelbrot.wav
```

There, that was painless. "-vn" told ffmpeg not to bother with any video, and the "-acodec", "-ar" and "-ac" options ensured the data was compatible with a CDDA.

If we look more closely at the output of ffmpeg, we see that the audio is MP3-encoded in the movie file:

```
Input #0, flv, from 'mandelbrot.flv':
  Duration: 00:04:25.7, start: 0.000000, bitrate: N/A
  Stream #0.0: Audio: mp3, 22050 Hz, mono
  Stream #0.1: Video: flv, yuv420p, 320x240, 29.97 fps(r)
```

Surely it must be possible to simply extract this audio data without processing it in any way so that I can upload it to my MP3-player and listen to it on that, right?

```
$ ffmpeg -i mandelbrot.flv -vn -acodec copy mandelbrot.mp3
```

The audio codec "copy" tells ffmpeg to use the same codec for encoding as was used for decoding. Also, since we're not overriding any of the other audio parameters such as sample rate or bit rate, we should be retrieving the sound track exactly as it was multiplexed into the movie file.

There is a similar "-vcodec copy" option that applies to the video stream.

It is also possible to extract the sound track from a DVD-Video. If you want to go directly from the DVD to audio data you can use MPlayer and its "-ao pcm:file=output.wav" option to get a WAV file with a sampling rate of 48000 Hz and 16-bit, stereo samples that you can process as you wish. However, if you've used a tool such as tccat to obtain a .vob file of a DVD title, you can feed the .vob file to ffmpeg and extract the audio that way. This has the advantage that you can manipulate the audio (resample, compress, downmix etc.) on-the-fly using the appropriate -acodec.

MAPPING CHANNELS

Coming back to the .vob file ripped from a DVD with tccat, it's commonplace for such multimedia files to have multiple audio streams embedded in them. Indeed, the DVD standard provides for up to 8 audio streams. Unless instructed otherwise, ffmpeg will operate on the first available sound track. It so happens that I have such a .vob file on-disk, so let's see what ffmpeg thinks of it:

```
$ ffmpeg -i mr.vob
FFmpeg version SVN-r9607, Copyright (c) 2000-2007 Fabrice Bellard, et al.
{snipped}
Seems that stream 0 comes from film source: 25.00 (25025/1001) -> 25.00 (25/1)
Input #0, mpeg, from 'mr.vob':
  Duration: 00:03:16.2, start: 620.890956, bitrate: 7704 kb/s
  Stream #0.0[0x1e0]: Video: mpeg2video, yuv420p, 720x576, 6799 kb/s, 25.00 fps(r)
  Stream #0.1[0x89]: Audio: dts, 48000 Hz, stereo, 768 kb/s
  Stream #0.2[0x80]: Audio: ac3, 48000 Hz, 5:1, 384 kb/s
  Stream #0.3[0x83]: Audio: ac3, 48000 Hz, stereo, 96 kb/s
  Stream #0.4[0x82]: Audio: ac3, 48000 Hz, stereo, 96 kb/s
  Stream #0.5[0x84]: Audio: ac3, 48000 Hz, stereo, 192 kb/s
  Stream #0.6[0x2d]: Subtitle: dvdsub
  Stream #0.7[0x2c]: Subtitle: dvdsub
  Stream #0.8[0x2b]: Subtitle: dvdsub
  Stream #0.9[0x2a]: Subtitle: dvdsub
  Stream #0.10[0x29]: Subtitle: dvdsub
  Stream #0.11[0x28]: Subtitle: dvdsub
```

```
  Stream #0.12[0x27]: Subtitle: dvdsub
  Stream #0.13[0x26]: Subtitle: dvdsub
  Stream #0.14[0x25]: Subtitle: dvdsub
  Stream #0.15[0x24]: Subtitle: dvdsub
  Stream #0.16[0x23]: Subtitle: dvdsub
  Stream #0.17[0x22]: Subtitle: dvdsub
  Stream #0.18[0x21]: Subtitle: dvdsub
  Stream #0.19[0x20]: Subtitle: dvdsub
```

The first stream, #0.0, is the video stream. Stream #0.1 is the DTS-encoded sound track and #0.2 is its AC3-encoded Dolby 5:1 equivalent. Stereo audio streams #0.3 through #0.5 are soundtracks with commentaries. Say I want to create a mono MP3 from this with the commentary from the third audio stream, #0.3. If I don't tell ffmpeg which one to use, it'll go ahead and transcode the first one it finds, the DTS stream in this case. I don't want that. This is where the "-map" option comes in handy:

```
$ ffmpeg -i mr.vob -map 0:3 -vn -acodec mp3 -ar 22050 -ab 96k -ac 1 mr.mp3
```

"-map *input*:*stream*" tells ffmpeg to process the given stream. As we'll see later on, ffmpeg can process input from several files. "*input*" is the zero-based index of the input file we want to use – 0 for the first, 1 for the second etc. "*stream*" is the number of the stream within this file that we want to use, also zero-based. "-map 0:3" therefore means that we want to use the fourth stream in the first (and only, in this case) input file.

"-map" can also be used to create a new movie from this .vob file using, for example stream #0.0 for the video and #0.5 for the audio. If any streams in a video file are mapped with "-map" then they must all be specified explicitly. In this case the first "-map" option specifies the stream to use for the video and the second one specifies which stream to use for the audio:

```
$ ffmpeg -i mr.vob -map 0:0 -map 0:5 -vcodec mpeg4 -b 1000k \
-s 640x360 -acodec mp3 -ar 22050 -ab 64k -ac 1 -f avi mr.avi
```

MULTIPLE SOURCES

One of the pieces of equipment adorning my video rig here at home is a DVD recorder. Almost invariably, I record direct onto a DVD+RW so that I can take the program I'm recording apart, rework the audio track (boost the volume level among other things), put it back together again, archive the modified program onto a DVD±R and put the DVD+RW back into circulation for the next recording.

Once the audio track has been extracted and reworked, I can reassemble the movie in either of two manners:

1. Also extract the mpeg2video data from the .vob file and then multiplex it and the reworked audio (duly converted to MP2 or AC3) with mplex, or
2. Ask ffmpeg to pull the video in from the original .vob file and the audio from the reworked .wav audio file and transcode it on-the-fly.

Solution 1 is already the object of another howto page (which needs finishing now that I think of it). This is how we use solution 2:

```
$ ffmpeg -i oldmovie.vob -i altered_audio.wav -map 0:0 -map 1:0 -target ntsc-dvd \
-b required_video_bit_rate -aspect 16:9 newmovie.mpg
```

Or, if you'd rather use MP2 audio and a lower audio bit rate:

```
$ ffmpeg -i oldmovie.vob -i altered_audio.wav -map 0:0 -map 1:0 -target ntsc-dvd \
-b required_video_bit_rate -acodec mp2 -ab audio_bit_rate -aspect 16:9 newmovie.mpg
```

Obviously replace "16:9" with "4:3" if you're reworking a 4:3 aspect ratio movie. Also, this assumes that the first stream in the .vob file is the video stream, so you'll need to adjust the "-map 0:0" accordingly if, for example the video stream is the *second* stream as is the case with my DVD recorder, in which case you'll need "-map 0:1" instead. Either way round, this is the stream that'll be used for the video in the output file. The audio stream is mapped to "-map 1:0". The "1" means "second file" (remember, the list is zero-based) and the ":0" means "first stream".

DELAYING THE AUDIO OR THE VIDEO

Depending on how your original files were generated, it could happen that the sound and the picture become out of sync when you start processing them with ffmpeg. Thankfully, there's a mechanism that will allow you to stagger the audio and video with respect to each other in the output, thus bringing them back into alignment – the "-itsoffset" option, which is used this way:

```
$ ffmpeg -i input_1 -itsoffset 00:00:03.5 -i input_2 ...........
```

In this example, *input_2* will be delayed by 3.5 seconds. In other words, The content of *input_1* will start at the beginning of the movie generated by ffmpeg, and the content in *input_2* will start 3.5 seconds into the movie.

In real life you're unlikely to come across such an extreme case of A/V skew. I do, however, come across cases regularly that need correcting by a fraction of a second. Using MPlayer to fast-forward into the movie you're generating and then the + and - keys to add more or less delay in 100ms increments gives you a good idea of how much delay to use and whether it's the sound or the picture that needs delaying.

Remember to put the source that needs delaying **after** the "-itsoffset" option. Thus, if the audio comes in too soon and needs delaying:

```
$ ffmpeg -i video_source -itsoffset delay -i audio_source -map 0:x -map 1:y ......
```

Conversely, if the audio comes in too late and the video therefore needs delaying:

```
$ ffmpeg -i audio_source -itsoffset delay -i video_source -map 1:x -map 0:y ......
```

Note how the "-map" options specify video from the second source and audio from the first in this second example.

Note also that *video_source* and *audio_source* can be the same file. Nothing's to stop you from fixing (or breaking) the A/V sync within the same original movie.

WORKING WITH ANAMORPHIC VIDEO

This is where things start getting moderately scary. Up until now we've been playing with videos that weren't anamorphic to start with, meaning that the physical image displayed on-screen had the same aspect ratio as the number of pixels making up the width and height of the image. For example, we've converted a Flash Video file that was 320x240 pixels in size and had a physical aspect ratio of 4:3. 320/240 = 4/3, therefore there is no anamorphism. This time we're going to crop the sides off a 16:9 video from a DVD-Video in order to bring it back to 4:3.

First, we're going to have to come to grips with a concept that I call the "Anamorphic Factor", which describes how an image has to be stretched or compressed to go from its pixel aspect ratio to its physical aspect ratio.

It was pointed out earlier that all movies on an NTSC DVD-Video have a resolution of 720x480 pixels. The pixel aspect ratio is therefore 720/480. The physical aspect ratio of the 16:9 image on-screen is 16/9. If we divide the physical aspect ratio by the pixel aspect ratio we get the Anamorphic Factor, that I call delta **δ**, and which basically represents the aspect ratio of the pixels (the pixels themselves are not square in an anamorphic image).

There are three possible cases:

- δ > 1 – The image is stretched horizontally from its pixel aspect ratio in order to attain the desired physical aspect ratio.
- δ = 1 – The pixel aspect ratio and the physical aspect ratio are the same. There is no anamorphism.
- δ < 1 – The image is squeezed horizontally from its pixel aspect ratio in order to attain its physical aspect ratio.

In the present case: δ = 16/9 / (720/480)
δ = 16/9 x 480/720 = (16x480)/(9x720) = 7680/6480 = (2^9 x 3 x 5) / (2^4 x 3^4 x 5) = 2^5 / 3^3
δ = 32/27

Remember, we're about to cut the sides off our image rather than squeeze it, meaning that we need to maintain its anamorphic factor. So, how many pixels should we shave off the sides? Or, put another way, how many pixels wide would an image have to be so that, once the anamorphic factor δ is applied to it, it has a physical aspect ratio of 4:3?

Written mathematically, we're looking for "W" in this equation:

W/H x δ = 4/3, or
W = 4/3 x H/δ

We happen to know H, it's 480 pixels. Substituting the known values in the equation above, we get **W=540 pixels**

So, if we shave 180 pixels off our anamorphic image, reducing its width from its original 720 pixels to 540, we should end up with a new image that, when the same anamorphic factor, δ, is applied to it, should have an aspect ratio of 4:3. Let's check:

540/480 x δ = 540/480 x 32/27 = (540 x 32) / (480 x 27) = 17280 / 12960 = 2^7x3^3x5 / (2^5x3^4x5) = 4/3
QED.

This means that, in order to crop a 16:9 NTSC video and turn it into a 4:3 video, we'd have to do something like this:

```
$ ffmpeg -i original_video -cropleft 90 -cropright 90 \
-target ntsc-dvd ........ -aspect 4:3 output_video
```

This time, instead of cropping the sides of the image, let's add black bars at the bottom and the top (letterbox) in order to obtain the 4/3 aspect ratio that way. For this we're going to need to know the anamorphic factor of a 4:3 image on a DVD-Video, and we'll call it δ'.

δ' = 4/3 / (720/480) = (480 x 4) / (720 x 3) = 160 / 180 = **8/9**

Yes, a 4:3 video on an NTSC DVD has an anamorphic factor less than 1, meaning that the picture is squeezed horizontally before being displayed.

If we divide the height of our anamorphic 16:9 image by δ we'll know how many pixels high it would have to be in order to no longer be anamorphic. If we then multiply that height by δ' we'll know how high it'll be in our new 4:3 image (also anamorphic). From that we'll be able to deduce how many pixels thick the "black bar sandwich" will have to be. So, the new height, H', will be:

H' = (H / δ) x δ' = (480 / (32/27)) x 8/9 = 15 x 27 x 8/9 = 3240/9 = **360**

Let's check this. If the answer is correct, then an image of 720x360 should be 16:9 once adjusted by an anamorphic factor of δ', 8/9:

720/360 x 8/9 = 5760/3240 = (2^7 x 3^2 x 5) / (2^3 x 3^4 x 5) = 2^4 / 3^2 = 16/9
QED.

If our letterboxed image is 360 pixels high we're going to have to add another 120 pixels to make up the whole 720x480 image, so this is what the command line would look like:

```
$ ffmpeg -i input_video -target ntsc-dvd ............. \
-s 720x360 -padtop 60 -padbottom 60 -padcolor 000000 -aspect 4:3 output_video
```

Let's do the same thing but for a .3gp movie file to be played back on a 3G cellphone. The video codec we'll be using is the H263 codec, which only allows a few predefined picture sizes. We'll be using 176x144. However, .3gp movies are reproduced at an aspect ratio of 4:3, meaning that they, too, are anamorphic. The audio codec we'll be using is the AMR narrowband codec, which requires a sampling rate of 8000 Hz, mono audio and a bit rate no higher than 12 kbit/s. We're also going to reduce the frame rate to 10 frames per second.

We're dealing with a different output technology and therefore a different δ'. For this technology:

δ' = 4/3 / (176/144) = (4 x 144) / (3 x 176) = 576/528 = (2^6 x 3^2) / (2^4 x 3 x 11) = (2^2 x 3) / 11 = **12/11**

If our image is to be squeezed into something that's only 176x144 pixels in size, we know its new width but we need to calculate its new height. If the image wasn't anamorphic, that new height would be 176 / (16/9), or (176x9)/16 = **99** pixels. But it *is* anamorphic, so we need to multiply 99 pixels by the δ' for this device:

H = 99 x 12/11 = 9x12 = **108 pixels**

In order to make up the 144 pixels of height we'll need to add 36 pixels of black, that's 18 either side. So, the command line is going to look like

this:

```
$ ffmpeg -i input_video -vcodec h263 -s 176x108 -b 300k -r 10 \
-padtop 18 -padbottom 18 -padcolor 000000 \
-acodec libamr_nb -ar 8000 -ab 12.2k -ac 1 \
-f 3gp -aspect 4:3 output.3gp
```

TIPS & TRICKS

Most standard image resolutions in videos are multiples of 4. Now, when calculating new widths and heights, you're not always going to be lucky like we were in the examples of this howto, and you could quite conceivably find yourself in a situation where you have to shrink something to 462.625 pixels in height, for example. Obviously, you can't leave or crop fractions of a pixel, so you have to compromise. If you're just resizing something and not cropping or padding then round the results you get to the nearest multiple of 2. If, on the other hand, you're planning on cropping or padding, I suggest you round the values you find to the nearest multiple of 4. The inaccuracy in the final aspect ratio of your movie file will be imperceptible, but you'll have engineered things so that you can crop or pad an even number of pixels either side, thus keeping everything nice and centered. This said, most modern video codecs work in a way that that is far more efficient and less wasteful if both the horizontal and vertical pixel sizes are multiples of 16.

Calculating the video bitrate you should use when transcoding a movie can be tricky at times, but only when you go about it the wrong way. Start by deciding how big you want (or can allow) the files to be and divide that space by the duration of your movie. This will provide you directly with the combined audio and video bit rate that you can use. For example, what bitrates should be used when recording 120 minutes of video onto a DVD? Allowing for multiplexing and filesystem overhead and for a small safety margin, we can say that we have 4.5 billion (4,500,000,000, that's 45 followed by 8 zeros – got that, Scott?) bytes to play with. Divide that by 120 and we get 37,500,000 bytes per minute. Divide that by 60 and we get 625,000 bytes per second, or 5,000,000 bits (5,000 kbits) per second. Subtract from that the audio bit rate and you have the video bit rate you can use. Lower the resulting value by 5% to allow ffmpeg room to breathe because the effective bit rate achieved is variable and will never be precisely what you ask for.

You can also use ffmpeg to concatenate multiple videos into one long video. Start by transcoding all the individual videos into MPEG format, all with exactly the same bit rates, codecs, image resolutions, frame rates etc. Mistakes can be avoided by using one of ffmpeg's predefined targets such as ntsc-dvd or pal-dvd. Once that's done, simply string the resulting .mpg files together using "cat" and redirect the output to another .mpg file. Now, the timestamps inside the resulting, big .mpg file are all going to be messed up, so you'll have to process the big .mpg file with ffmpeg again. This will have the effect of putting the timestamps right.

NAVIGATION:
>> /

Last update: 01-FEB-2012
This page has been served 981371 times since 10-OCT-2006

Guide to Anamorphic Encoding in HandBrake

Short visual explanation

These are scaled down 50% in size, in case you couldn't tell.

Here's the size of a movie stored on a DVD. On the disc, the film is distorted. Instead of being truly widescreen, it's squeezed into a narrower frame.

To restore the proper shape, you can either squeeze the picture vertically or stretch the picture horizontally. One shrinks the image, and one expands it.

Here's the size of a movie that's been squeezed vertically. This shrinks the image. Notice how the width stays the same as what's stored on the DVD, but the height is reduced:

Here's the display size of a movie that's been stretched horizontally -- HandBrake's default. This expands the image. Note that it preserves the full height of the image stored on the DVD:

Anamorphic in HandBrake means encoding that distorted image stored on the DVD, but telling the video player how to stretch it out when you watch it. This produces that nice, big, widescreen image.

Enabling Anamorphic PAR

You enable anamorphic encoding by selecting Strict or Loose from the Anamorphic PAR menu in HandBrake's picture settings.

Strict is more precise, but Loose is more efficient and flexible. Both will give you that big, widescreen image.

Anamorphic DVDs

An image is stored on a DVD at 720*480 (NTSC) or 720*576 (for PAL). I'm in North America so I'll be using NTSC numbers in my examples. (See the PAL appendix for the differences.)

This is what the image stored on an anamorphic NTSC DVD looks like (for information on non-anamorphic DVDs, see the Hard Letterboxing Appendix):

Notice how it's distorted? That's the anamorphic part. DVDs are stored with a 1.5:1 aspect ratio.

Aspect Ratios

Wait, aspect ratio? What?

Aspect ratio is the width, divided by the height. DVDs are stored with a 720*480 resolution, and 720 / 480 = 1.5. That means it's 1.5 times wider than it is tall.

But video isn't meant to be seen with a 1.5:1 aspect ratio. It's not wide enough for movies and widescreen TV, and too wide for standard TV. One common widescreen aspect ratio is 1.78:1, or 16/9. This is the native aspect ratio of widescreen TVs. Standard TVs use an aspect ratio of 4/3 (1.33:1).

DVD video has an "aspect ratio flag" which tells the DVD player how to distort the picture stored on the DVD to recreate the original film aspect ratio. This flag is either 4/3 or 16/9.

Of course, a lot of movies are wider than 16/9. One popular aspect ratio is 2.35:1, which is quite a bit wider. When this is stored on a DVD, it too is given the 16:9 aspect ratio flag. Of course, the content itself is wider. To cope with this, black bars (letterboxing) line the top and bottom of the picture. This can be a difficult part to grasp. At least, it was for me. So I'm going to repeat this with a few different wordings.

Letterboxing

When anamorphic DVD content is stretched for viewing on a widescreen TV, it *always* stretches to 854*480. 480 is too "tall" for films wider than 16:9. To make it "shorter" the top and bottom are matted with black lines. So whether a movie has an aspect ratio of 1.78:1, 1.85:1, or 2.35:1, it is always going to displayed 854 pixels wide. The visible frame height (when you remove the letterboxing) just gets shorter and shorter to match the aspect ratio—480, 460, and 360, consecutively. (Those height and width values are only approximate. For details, see the Macroblock Appendix.)

The storage width is the width of the visible frame on the DVD (almost always 720) and the display width is 854 (storage height of 480 times 16/9). The display height is the height of the visible frame on the DVD—the frame after cropping away black bars. This is going to be roughly equivalent to the display width divided by the film's aspect ratio.

What happens on a TV

When you play the DVD on a widescreen 16:9 television, it keeps the height and stretches the width (854*480). This is what it means when a DVD box proclaims "Enhanced for widescreen." When you play the DVD on a standard 4:3 television, it reduces the width to 640 (the maximum width for a standard TV) and squishes the height to 360 to match the aspect ratio.

Both ways recreate the film's aspect ratio. The first way multiplies the height (480) by a 16:9 aspect ratio, and uses that for the width. The second method takes the width (720), reduces it to the maximum width of a standard TV (640), divides it by the film's aspect ratio (1.78:1 aka 16:9), and uses that as the height.

Flexibility

To put this another way, "anamorphic" means the movie doesn't have a single, native shape that you watch. Instead, it shape-shifts. If you're watching it on a standard TV, it morphs to fit it. When you play it on a widescreen TV, it morphs to fit it, too.

What happens when anamorphic is turned off

When Anamorphic is disabled, HandBrake corrects the aspect ratio by maintaining the width and squishing the height to match. The result looks like this:

This is very similar to what happens when you play a DVD on a standard 4:3 television. The only difference is, HandBrake leaves the width at 720, instead of reducing it to 640. So it divides 720 by the aspect ratio, giving you output dimensions of 720*404. (If you're testing me on this and you get a width of 704 or a height of 400, please see the Macroblock Appendix.)

Of course, that means you're reducing the vertical line count from 480p to 404p... a significant reduction in picture quality.

There are other ways to go about encoding the anamorphic video, in order to keep that from happening.

Anamorphic encoding methods

Crude anamorphic

The crude way (which is easily done from !HandBrakeCLI), is to maintain the height and "hard" stretch the frame width to 854 (for details, see the CLI Appendix). The problem with this is that you end up storing an image that's got a higher resolution than your DVD source—it takes up more space. Instead of a 720*480 frame or a 720*404 frame, you're writing a 854*480 frame to disk.

Strict anamorphic

The second method is far less wasteful. Why not do the same thing a DVD does? Store in one aspect and display in another.

With this method, you can preserve the full frame on the DVD, without having to store it at its wider display resolution. Compared to storing 854*480 on disk, storing 720*480 reduces file size while maintaining exactly the same quality.

Now, how do we go about doing this?

Vide on a computer is stored in a container file, be it .mp4 or .mkv or .avi or .ogm or something else. Inside that container are tracks or streams. Usually, there will be one video track and one audio track.

The smart way of handling anamorphic is to store the display information in the video track. And this is exactly what HandBrake does:

This technique is so clever, VLC doesn't even realize what's going on. It thinks it's displaying 720*480 when it's really showing 854*480.

Pixel Aspect Ratio

In fact, it *is* displaying 720*480. Only, the video track is telling VLC: "Display this with wide pixels instead of the square ones you usually use." So instead of an image of square blocks, it becomes an image of wide rectangular blocks.

Because computers think of video in terms of square pixels, VLC has to figure out what arrangement of square pixels is needed to reproduce the image in its correct dimensions. It does this by multiplying the storage width (720) by a ratio: the Pixel Aspect Ratio, or PAR. By default, the PAR is 1:1. With that ratio, what you see is what you get—square pixels. The video is stored and displayed with the same dimensions. In order to recreate 16:9 pixels from 1:1 pixels, the ratio is 32/27 (16*2 / 9*3). For every 32 square pixels across, it uses 27 square pixels up and down. You already know this, another way: it produces the same results as multiplying the height (480) by 16/9. Multiply 720 by 32 and divide that by 27, and you end up with 854, the display width in square pixels.

This is anamorphic PAR, and it is very, very sexy.

The downside of strict anamorphic

Strict anamorphic concentrates on one thing and one thing only: preserving the exact visible frame of a DVD, displayed to exactly the same size as it would be from the DVD.

This means it will sometimes use odd dimensions, ones that don't divide cleanly by 16. When this happens, the video encoders cannot work as efficiently — x264 warns that "compression will suffer."

It also means that, when using strict anamorphic, it is impossible to change the stored size of the encoded frame. It will simply use the exact frame size of the DVD and apply cropping.

Loose anamorphic

Loose anamorphic starts off the same way as strict -- with the exact visible frame on the DVD. But then it adjusts the dimensions to be sure they divide cleanly by 16. After that, it adjusts the display size so the film's aspect ratio is preserved with the new dimensions.

You can also scale the width of the storage frame, using loose anamorphic. For example, the full-sized storage frame has a width of 720. You could scale that down to 640. HandBrake will automatically keep the aspect ratio of the storage frame. So as the full-sized is 720*480 (a 1.5:1 aspect ratio), a scaled down one would be 640*432 (as close as one can get to 1.5:1 while keeping dimensions that divide cleanly by 16). HandBrake will then calculate the proper display size for that scaled frame, one which preserves the source's film aspect ratio.

There are some other minor differences in the output of loose versus strict, but they are nerdy. Only the curious and sleepless need bother reading about them in the ITU appendix.

What about QuickTime? And iPods? And AppleTVs? And iPhones?

In order to preset anamorphic video in QuickTime, HandBrake adds extra display information to the .mp4 container wrapping the video, called a picture aspect atom.

This is different from the information in the video stream, but achieves the same effect.

In this manner, HandBrake creates anamorphic video that can be displayed with QuickTime, as well as with iPods, as well as with iPhones, as well as with AppleTVs, as well as with open source video players like VLC and MPlayer.

Other Resources

- http://c133.org/blog/tech/screeching_handbrake.html (the birth of the feature)
- http://seemoredigital.net/03_Video_Only_Info/What_is_an_anamorphic_encode.html
- http://www.3ivx.com/support/par.html
- http://lipas.uwasa.fi/~f76998/video/conversion/
- http://en.wikipedia.org/wiki/Anamorphic_widescreen
- http://gregl.net/videophile/anamorphic.htm
- http://www.dvdfile.com/news/special_report/production_a_z/anamorphic.htm

---

Appendices

CLI Appendix

Hard stretching to anamorphic in !HandBrakeCLI

It's really easy. You just have to specify the visible frame height. Say your command is:

```
./!HandBrakeCLI -i dvd -o movie.mp4
```

What you add for anamorphic encoding depends on the film's aspect ratio.

- 1.78:1 means adding "-l 480" to the end of the command.
- 1.85:1 means adding "-l 460"
- 2.35:1 means adding "-l 360"

So for a 1.85:1 movie your complete command would be:

```
./!HandBrakeCLI -i dvd -o movie.mp4 -l 460
```

What this is going to do is give you output dimensions of 854*460. That means you are storing a movie with a frame size 33% larger than the DVD. So only use it to play around. It's a total waste of storage space.

Instead do...

Anamorphic PAR in !HandBrakeCLI

This is even easier. For strict anamorphic just add a "-p" like so:

```
./!HandBrakeCLI -i dvd -o movie.mp4 -p
```

And for loose anamorphic, use a "-P":

```
./!HandBrakeCLI -i dvd -o movie.mp4 -P
```

That will output a movie with dimensions of 720*480 (for a 1.78:1 NTSC film) but which QuickTime, VLC, and MPlayer will display as 854*480. The same display as a hard stretched anamorphic DVD, without storing a 33% larger image.

Note that when using loose anamorphic, you can also include a storage width to use. In addition, you can choose what number the dimensions should divide by cleanly. By default, this is 16, the size of MPEG macroblocks. It is passed as the optional argument to -P.

So, for example, to do an anamorphic encode to a frame sized 640*424, you could use:

```
./!HandBrakeCLI -i dvd -o movie.mp4 -w 640 -P=8
```

Hard Letterboxing Appendix

Some widescreen DVDs, especially older ones, aren't anamorphic. These DVDs are specifically designed to play on standard 4:3 TVs. The "aspect ratio flag" for these discs is set to 4:3 even though the content's aspect ratio is wider.

When you play it on a standard square TV, everything's shiny. However, when you play it on a widescreen TV, you will have black bars not only on the top and bottom, but also on the sides. This gives you a far smaller picture. Those side-bars are often called "pillars." This technique is often used in broadcast television to deliver 16/9 content to standard definition sets. Because it gives you an even smaller picture than letterboxing does, it's sometimes called postage-stamping.

Macroblock Appendix

MPEG video encoders (like the MPEG-2 used on DVDs or the MPEG-4 variants used by HandBrake) work by dividing the video frame into blocks. The entire frame becomes a grid of blocks, 16 pixels high and 16 pixels wide. These 16x16 blocks are called macroblocks. When you encode video, you have to use height and width values that are multiples of 16. When the height or width doesn't divide cleanly into 16 (that is to say, when there is a remainder), the video encoder has to make up extra "garbage" information for the edges of the frame. This increases the file size or decreases the video quality, depending on whether you're targeting a constant quality or a size/bitrate.

An unhappy side effect of this is that aspect ratios can only be approximate. 720*360 is the storage dimensions of a 2.35:1 movie after cropping. But because 360 is not divisible by 16, you have to bump up to 368 or down to 352. Suddenly the display aspect ratio is 2.32:1.

This is why sometimes, using HandBrake with anamorphic encoding disabled, you will get an output width of 704 for a 1.78:1 film. For whatever reason, HandBrake has decided it needs a width that is slightly lower than 720. But in order to honor macroblocking requirements, widths have to be divisible by 16. 704 is next possible width lower than 720.

Similarly, even though the "perfect" height for a movie 720 pixels wide with a 1.78:1 aspect ratio would be 404, that number isn't cleanly divisible by 16, while 400 is.

When using "strict" anamorphic, HandBrake ignores macroblocking requirements for anamorphic encoding. It makes perfectly preserving the visible frame with a precise aspect ratio its top priority.

When using loose anamorphic, HandBrake uses dimensions that divide cleanly by 16. Using the CLI, you can also tell it to use a different number. Sometimes using 8 instead of 16 allows dimensions that are far closer to the correct aspect ratio.

PAL Appendix

PAL storage frame size is 720*576. That means the display width is always the height times 1.78 (16/9), or 1024. Both the storage and the display height are 1024 divided by the aspect ratio.

- PAL output dimensions:
  - 1.78:1: 1024*576
  - 1.85:1: 1024*554
  - 2.35:1: 1024*436
- PAL storage dimensions:

- 1.78:1: 720*576
- 1.85:1: 720*554
- 2.35:1: 720*436

ITU pixel aspects

The standard pixel aspect ratios for NTSC video are 32/27 for widescreen and 8/9 for fullscreen.

That means if a frame is 720 pixels wide, it will be displayed at a width of 720 * 32 / 27, or 853.333... if it's widescreen, and it will be displayed at 720 * 8 / 9, or 640 pixels wide if it's fullscreen.

But this depends on the video actually using all 720 pixels of width. A lot of material that was originally prepared for analog broadcast (a lot of older stuff) doesn't use the full width. They knew that old analog television sets used overscan. That means they would zoom in on the image, cropping away the outer image. This is all ITU specification.

Since viewers would never see it, any visible detail they put there would be lost. Instead, they matte the left and right sides with thin black bars, each around 8 pixels wide. This reduces the width from 720 to 704. As a result, a different pixel aspect ratio is used.

After all, 704 * 32 / 27 is only 834.37 pixels! The whole image is still there in the frame, they just used a slightly different aspect ratio to squeeze it in. Mirroring that, to display it properly, you have to use a slightly different aspect ratio when stretching it out.

Fortunately, there's no guessing involved. It's all laid out by MPEG (Yes, the ITU PAR numbers !HandBrake uses come from MPEG not ITU. This is because they are very similar, and better for use with MPEG-4 video like HandBrake produces. So when you read in the following lines something like "The ITU PAR numbers for foo are..." it really means, "The MPEG-4 PAR numbers equivalent to the PAR values established in ITU 601-R BT.601 spec are...").

The ITU PAR numbers for NTSC are 40/33 for widescreen and 10/11. And indeed, 704 * 40 / 33 = 853.333, and 704 * 10 /11 = 640.

For PAL, the ITU PAR numbers are 16/11 for widescreen and 12/11 for fullscreen. 704 * 16 / 11 = 1024, 704 * 12 / 11 = 768, so there too the results are the same as they would be using the normal pixel aspect ratios and a full 720 pixel wide frame.

When using loose anamorphic, HandBrake will use the ITU pixel aspects whenever it crops more than 14 pixels from the width. This applies whether it is detecting the cropping itself using autocropping, or custom cropping from user input.

Last modified 5 years ago

Attachments (3)

- fireflydefault.png (403.5 KB) - added by *jbrjake* 5 years ago. "HandBrake's old default output"
- fireflypar.png (609.9 KB) - added by *jbrjake* 5 years ago. "Anamorphic display"
- fireflysar.png (432.2 KB) - added by *jbrjake* 5 years ago. "Right dimensions this time."

Download all attachments as: .zip

Download in other formats:
Plain Text

#309 closed defect (invalid)

Opened 3 years ago
Closed 3 years ago
Last modified 3 years ago

width or height not divisible by 2

Reported by:	AlanJames1987	Owned by:	
Priority:	normal	Component:	undetermined
Version:	git-master	Keywords:	
Cc:		Blocked By:	
Blocking:		Reproduced by developer:	yes
Analyzed by developer:	yes		

Description

While using -vf 'scale=' if the output has an a height that is an odd number it gives an error like below.

[libx264 @ 0x175c880] width or height not divisible by 2 (854x363)

This has caused me lots of headache when down-converting large amounts of video and not having all of it be converted.

Change History (7)

Oldest first / Newest first
Comments only

Changed 3 years ago by cehoyos — comment:1

- **Component** changed from *avfilter* to *undetermined*
- **Status** changed from *new* to *open*

Command line and complete, uncut output missing (but I fear you used the wrong bug tracker).

Changed 3 years ago by AlanJames1987 — comment:2

Command line

```
#!/bin/bash
ffmpeg -i input.mov \
  -vcodec libx264 -vf 'scale=854:-1' -b 854k \
  -acodec aac -strict experimental -ab 112k \
  -vpre baseline -threads 0 -map_metadata -1 -y output.mp4
```

uncut output

```
ffmpeg version N-31018-gebc64dc, Copyright (c) 2000-2011 the FFmpeg developers
  built on Jun 26 2011 04:42:51 with gcc 4.5.2
  configuration: --enable-gpl --enable-version3 --enable-nonfree --enable-postproc
  libavutil    51. 10. 0 / 51. 10. 0
  libavcodec   53.  7. 0 / 53.  7. 0
  libavformat  53.  4. 0 / 53.  4. 0
  libavdevice  53.  1. 1 / 53.  1. 1
  libavfilter   2. 23. 0 /  2. 23. 0
  libswscale    2.  0. 0 /  2.  0. 0
  libpostproc  51.  2. 0 / 51.  2. 0
[mov,mp4,m4a,3gp,3g2,mj2 @ 0x21d2440] Unknown container channel layout.
[mov,mp4,m4a,3gp,3g2,mj2 @ 0x21d2440] max_analyze_duration 5000000 reached at 5013

Seems stream 0 codec frame rate differs from container frame rate: 5994.00 (5994/1
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'input.mov':
  Metadata:
    major_brand     : qt
    minor_version   : 537199360
    compatible_brands: qt
    creation_time   : 2008-07-15 16:22:37
    comment         : Encoded and delivered by apple.com/trailers/
    comment-eng     : Encoded and delivered by apple.com/trailers/
    copyright       : © 2008 Warner Bros. Pictures. All Rights Reserved
    copyright-eng   : © 2008 Warner Bros. Pictures. All Rights Reserved
    title           : The Dark Knight
```

```
    title-eng       : The Dark Knight
  Duration: 00:02:06.93, start: 0.000000, bitrate: 10488 kb/s
    Stream #0.0(eng): Video: h264 (Main), yuv420p, 1920x816, 10275 kb/s, 23.98 fps
    Metadata:
      creation_time   : 2008-07-15 16:22:37
    Stream #0.1(eng): Audio: aac, 48000 Hz, 5.1, s16, 208 kb/s
    Metadata:
      creation_time   : 2008-07-15 16:22:37
    Stream #0.2(eng): Data: tmcd / 0x64636D74
    Metadata:
      creation_time   : 2008-07-15 16:22:37
[buffer @ 0x21d6280] w:1920 h:816 pixfmt:yuv420p tb:1/1000000 sar:0/1 sws_param:
[scale @ 0x21d5260] w:1920 h:816 fmt:yuv420p -> w:854 h:363 fmt:yuv420p flags:0x4
[libx264 @ 0x21d68c0] Default settings detected, using medium profile
[libx264 @ 0x21d68c0] width or height not divisible by 2 (854x363)
Output #0, mp4, to 'output.mp4':
    Stream #0.0(eng): Video: libx264, yuv420p, 854x363, q=2-31, 854 kb/s, 90k tbn,
    Metadata:
      creation_time   : 2008-07-15 16:22:37
    Stream #0.1(eng): Audio: libfaac, 48000 Hz, 5.1, s16, 112 kb/s
    Metadata:
      creation_time   : 2008-07-15 16:22:37
Stream mapping:
  Stream #0.0 -> #0.0
  Stream #0.1 -> #0.1
Error while opening encoder for output stream #0.0 - maybe incorrect parameters su
```

The problem is in avfilter somewhere in the vf_scale.c file.

Changed 3 years ago by saste — comment:3

- **Analyzed by developer** set
- **Reproduced by developer** set
- **Resolution** set to *invalid*
- **Status** changed from *open* to *closed*

This is a limitation of libx264, which is unrelated to libavfilter/scale.

When choosing scale=W:-1 the filter won't round the scaled H by 2, you need to explicitely express this to the scale filter, either by employing scripting or using expressions in the scale filter itself.

For example the following will round the scaled height to a multiple of 2, keeping the same input aspect ratio:
scale="854:trunc(ow/a/2)*2"

Changed 3 years ago by cehoyos — comment:4 follow-up: ↓ 5

Wouldn't scale="854:-2" be much simpler?

Changed 3 years ago by saste — comment:5 in reply to: ↑ 4

Replying to ~~cehoyos~~:

> Wouldn't scale="854:-2" be much simpler?

Patches are welcome, if you believe that is cleaner and if you can provide a consistent semantics for negative h/w values.

Changed 3 years ago by cehoyos — comment:6 follow-up: ↓ 7

```
If the value for @var{width} or @var{height} is -n (negative), the scale filter wi
use, for the respective output size, a value that maintains the aspect
ratio of the input image, and rounds to the nearest multiple of +n.
```

Do you believe anything is unclear or inconsistent?

Changed 3 years ago by saste — comment:7 in reply to: ↑ 6

Replying to ~~cehoyos~~:

> ```
> If the value for @var{width} or @var{height} is -n (negative), the scale filter w
> use, for the respective output size, a value that maintains the aspect
> ratio of the input image, and rounds to the nearest multiple of +n.
> ```
>
> Do you believe anything is unclear or inconsistent?

Looks sane.

Note: See TracTickets for help on using tickets.

Download in other formats:
Comma-delimited Text | Tab-delimited Text | RSS Feed

« Compare RED, Canon 5D and Lumix GH1 in 1 spot

HDSLR Cinema tool: frame rate conversion »

Convert between framerates (24 fps, 25 fps, 30 fps, ...)

Published on March 17, 2010 in Workflow. 5 Comments
Tags: ffmpeg, opensource.

The 5D originally only shot in 30 fps (NTSC-inspired, undoubtedly, since it was meant for USA news agencies anyway). So when the footage was to be used for TV (PAL in Europe = 25 fps) or for cinema (24 fps), it had to be converted. Also, when you shoot in 720p60 (at 60 fps) so that you can make a nice slow motion effect, you have to 'convert' your material. We already referred to an article about how to 'conform with Cinema Tools/FinalCutPro'. But let's talk some more about this conversion and how to do it with a free open-source tool like `ffmpeg`.

1. Conversion with fixed # of frames.

This is what is referred to as '*conforming*'. You keep the same number of frames, so a 20 second long 30 fps clip (20 x 30 = 600 frames) becomes 25 seconds at 24 fps. The same number of frames are played at a slower speed. The only thing you have to do for the video is to change the metadata of your clip. The audio will have to be played slower too, so will sound lower. If you want to fix this, you have to pitch this up (with audacity or sox).

How to do this with ffmpeg? Well, in two steps:

- extract the frames as rawvideo
  `ffmpeg -i input.mov -f rawvideo -b 50000000 -pix_fmt yuv420p -vcodec rawvideo -s 1920x1080 -y temp.raw`
- recreate the video with new framerate
  `ffmpeg -f rawvideo -b 50000000 -pix_fmt yuv420p -r 24 -s 1920x1080 -i temp.raw -y output.mov`
- To save on needed intermediate disk space, you could send the output of the first to stdout and pipe it to the input of the second

2. Conversion with fixed running length.

This method keeps the total length, but in order to do so, it has to *interpolate* (estimate frames that are between 2 original frames) . In other words, for each second you get 30 input frames and you need to create e.g. 24 output frames. Output frame 3 of that second will be created out of input frames 3 and 4, combined in some way. If this interpolation is done with a low quality/fast algorithm, any steady smooth movement in the input movie, might become jerky and unnatural. Therefor this procedure will be slow (might be slower than real time – you'll need more than 1 hour of conversion time for each hour of footage) The audio will remain untouched, since the total length does not change.

How to do this with ffmpeg? Easy:

- convert framerate in one step:
  `ffmpeg -i input.mov -sameq -r 24 -y output.mov`

Unfortunately, this is quite slow AND low quality. So you might want to look at tools like FinalCutPro to do this kind of conversion. Also, MVtools (AVIsynth) seems to be able to do this better, I still have to check that out.

Recent Posts
Nikon D800 vs Canon 5D Mk III
Dimensions: feature film shot on 5D
Digital cinema aspect ratios
Depth-of-field calculator: now with visualisation!
Composition calculator

HDSLR News
Black Magic Cinema Camera (BMCC)
Beautiful & Befuddling/Canon C100 leads to murky future for mid- to upper-range...
FilmmakerIQ.com
Samsung Threatens to Strand Indian Bloggers in Germany
NoFilmSchool
HDSLR Cinema News

Categories
Select Category

Ads

Archives
Select Month

Information
About
HDSLR Cameras
Tools

Movies
Beyond the Still
DSLR Cinema
DSLR Flix

News
Cinema 5D
EOS HD
NoFilmSchool
Planet5D

People
Byron Shah
Chase Jarvis
Dan Chung
F-Stop Academy
Kevin Shahinian
Oliver Peters
Philip Bloom
Rodney Charters
Shane Hurlbut
Stu Maschwitz
Tom Guilmette
Vincent Laforet

Tools
HDSLR Tools

5 Responses to "Convert between framerates (24 fps, 25 fps, 30 fps, …)"

TheToad
May 1, 2010 at 10:30 pm

Unfortunately the way how ffmpeg (also the ffmpeg broadcast version http://code.google.com/p/ffmbc/) read the h264 mov file is not very good: you get the highlight very compressed and you lose information if you want do a good grading!!!
Personally, after a long research, i use another way :
Using avisynth with QTinput plugin (to open mov file) and open the avs file with virtualdub or ffmpeg to export in your favorite format and frame rate.

if you use open avs file with ffmpeg you have to add flipvertical() command in avs file; that is not userfull if you use virtualdub(i don't know why)
I will post soon some where a small tutorial to how menage all this and also export in loss-less free codec (huffyuv or Lagarith)
sorry for my poor English
i hope this is user full

yellow
August 25, 2010 at 11:53 pm

@TheToad, I find QTInput scales the luma, even with latest QT Alternative installed. It appears to scale the luma twice though, so levels appear correct until you check them on a waveform or histogram to see combing and data loss.

So i use Haali Media splitter and remux into a matroska container, then use AVISynth & Haali bundled .dlls to read the mkv's to decode the h264.

Also for anyone reading, never use FFmpeg to convert to RGB always use AVISynth. FFmpegs swscale expands the luma crushing blacks and clipping whites, and on default settings for interpolation looses about 50% of colour data compared to AVISynth's conversion.

This all works on Linux too with Wine.

Ome Two
December 28, 2010 at 3:56 pm

I tried to "upframe" – convert mp4 with 23.976 to 30 frames rate. Does not work. I mean, output file still has 23.976 frames rate.

alienresident
March 19, 2012 at 11:12 pm

Found this post and it was very helpful start. I wanted to give back some additional info I discover for anyone who comes here via google.

1. This is how you pipe the output of step 1 into step 2 so you don't have to create (& then delete) the temp.raw file

ffmpeg -i input.mov -f rawvideo -b 50000000 -pix_fmt yuv420p -vcodec rawvideo -s 1920×1080 -y pipe:1 | ffmpeg -f rawvideo -b 50000000 -pix_fmt yuv420p -r 24 -s 1920×1080 -i – output.mov

Secondly, by default this will create a h264 file if you want to use a different codec such as Apple Pro Res (only in the newer builds of FFmpeg)

ffmpeg -i input.mov -f rawvideo -b 50000000 -pix_fmt yuv420p -vcodec rawvideo -s 1920×1080 -y pipe:1 | ffmpeg -f rawvideo -b 50000000 -pix_fmt yuv420p -r 24 -s 1920×1080 -i – -vcodec prores -profile 2 output.mov

serg_ulixes
June 8, 2012 at 10:02 am

I think I found a more elegant way to do that using "-vsync passthrough" option. It is

conforming my 59.97fps footage to slow motion 23.98 without frame skipping:

for %%a in ("*.mov") do ffmpeg -i "%%a" -s 1280×720 -sws_flags lanczos -r 23.98 -vsync passthrough -vcodec dnxhd -b 110M -mbd rd -acodec copy "%%~na.DNxHD.mov"

« Compare RED, Canon 5D and Lumix GH1 in 1 spot HDSLR Cinema tool: frame rate conversion »

HDSLR Cinema tool: frame rate conversion « HDSLR Cinema News
Pingback on Mar 29th, 2010 at 2:11 pm

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Submit

« Compare RED, Canon 5D and Lumix GH1 in 1 spot HDSLR Cinema tool: frame rate conversion »

How can I crop a 16:9 video to 4:3, cutting off the edges?

0

I've got a 720p video, that contains a picture that's only 4:3, so the edges left and right stay empty/black.

I want to cut of the left and right so that video is 4:3. How do I do this in Windows?

- Input: WMV, 1280x720px
- Output: WMV, AVI or MPEG, 960x720px

video-editing video-conversion

share | improve this question

edited Oct 7 '11 at 10:04
slhck ♦
71.6k 14 139 195

asked Oct 7 '11 at 10:00
Hinek
170 1 1 8

add comment

2 Answers

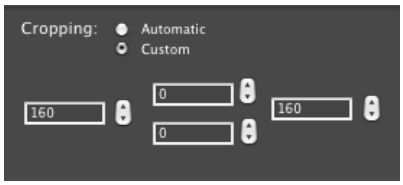
active oldest votes

4

Download Handbrake.

Open your source video, then select *Picture Settings*. The following is a screenshot from OS X, but it's similar on Windows.

Your source is 1280px wide. You need 960px, thus 320px to get rid of. This means you want to crop 160px at each side of the video.



Choose your export options as you like, but you should know that by doing this, you will lose a fair amount of video quality, as the video will be re-encoded. This is why you should probably use the highest quality settings possible (i.e. the *High Profile*, with h.264 as codec.

Also note that cropping and destructively editing a video because it was poorly encoded (i.e. with black borders, "pillarboxed") is probably the wrong approach. Consider switching to a playing software that allows you to selectively crop the video, such as VLC.

share | improve this answer

edited Oct 7 '11 at 10:14

answered Oct 7 '11 at 10:06
slhck ♦
71.6k 14 139 195

add comment

2

With a recent version of ffmpeg, you can achieve this effect from the command-line:

```
ffmpeg -i input.mp4 -filter:v 'crop=ih/3*4:ih' \
-c:v libx264 -crf 23 -preset verfast -c:a copy output.mp4
```

See the ffmpeg filter documentation for more information about the crop filter.

The `-crf 23 -preset veryfast` are libx264 options that set the quality. Basically: a lower CRF=better quality, but larger file. A faster preset means faster encoding, but a larger file. See this page on the excellent FFmpeg wiki for details of their use.

share | improve this answer

edited Aug 19 '13 at 16:11

answered Jan 17 '13 at 12:47
evilsoup
3,483 1 6 27

A little explanation would be fine. The docu is quite good but a pain to read. What is the meaning of `-crf 23`? – ManuelSchneid3r Aug 19 '13 at 13:12

add comment

tagged
video-conversion × 290
video-editing × 271

asked **2 years ago**
viewed **2064 times**
active **5 months ago**

Related

2 How can a trimmed avi file be three times larger than the original

1 Is there a tool which can resume video conversion?

1 How can I merge AVI movies at 16:10 aspect ratio and maintain the aspect ratio in the output?

3 How to cut at exact frames using ffmpeg?

2 How can I crop a video to a part of the view?

3 Ffmpeg sync error, first image showing really fast

3 Accurate and reliable movie editing with avconv

4 dividing a video clip by 4 areas

0 Cropping video to different parts with same bitrate in ffmpeg

0 ffmpeg - concat 2 mp4 files results in bad output file

Hot Network Questions

Why there is such a big difference between "Size" and "Size on disk"?

How do I explain to a 6 year old why people on the other side of the earth don't fall off?

The Spiral of Roots in TikZ

What is the difference between a point and a vector

Is it ethical to write answers to work-relevant Stack Exchange questions on the clock?

more hot questions

Your Answer

Post Your Answer

By posting your answer, you agree to the privacy policy and terms of service.

Not the answer you're looking for? Browse other questions tagged video-editing video-conversion or ask your own question.

question feed

reneVolution

Understanding FFmpeg – Part III: Cropping

Posted on: 23. May 2013
By: René Calles
With: 5 Comments

Home » Understanding FFmpeg – Part III: Cropping

If you've ever been in a situation where your recorded video had a blank interval or you just want to get a specific part of your video you'll definitely want to crop your video. In this article i'll show you how to do this with FFmpeg.

Example video

Ok, before you leave the keyboard and stop reading .. you will not need a ruler to fix this. Let's have look how we can do it the easy way using FFmpeg.

Get the cropping values

At first we need to find out the necessary values for the cropping filter. This can be done by FFplay using the cropdetect filter. This can be done using the command line `ffplay -i YourMovie.mp4 -vf "cropdetect=24:16:0"` or using FFmpeg with a command line like `ffmpeg -i YourMovie.mp4 -vf "cropdetect=24:16:0" dummy.avi` .

The cropdetect filter is configured with the arguments *cropdetect=limit:round:reset* with the following meaning:

limit = The threshold of the intensity of the black bar. You can play around with that, it worked mostly well for me with the default value 24

round = Ensures the output resolution is divisible by the value specified. 16 is best for most video codecs.

reset = Determines after how many frames cropdetect will reset the previously detected largest video area and start over to detect the current optimal crop area. This is especially useful when tv logos interrupt the video.

To output in your shell will look like this:

cropdetect output Shell

```
1 [Parsed_cropdetect_0 @ 0x7fe4f1
```

```
2 [Parsed_cropdetect_0 @ 0x7fe4f1
```

What's that ? Right, the last part on every line with "***crop=...***" are our cropping values to remove all black bars around our movie.

Crop it like it's hot

Now it's time to remove those annoying black bars around this movie using

```
Shell
1 ffmpeg -i YourMovie.mp4 -vf "crop
```

which will result in the expected output like this:

To finally explain the parameters used with the crop filter in a more general way, here we go:

The crop filter takes four arguments `crop=out_width:out_height:x:y` and can be explained as

out_width/output_height represents the cropped output width of the video.

x/y is the start coordinate from the top left corner.

In my example the cropped output width is 640px and the output height is 256px in total, starting from (0,36) coordinate.

Cropping with fixed values

Another approach is to crop your video to a fixed area. This can be done using regular expressions inside the crop filter configuration.

For e.g. to crop your video 32px on left/right and 16px on top/bottom use this configuration `crop=iw-64:ih-32:32:16`. The parameter ***iw-64*** is the cropped output width calculated by the input width – 64px as we wanted to crop 32px from each side. The same is done for the output height with ***ih-32*** which results in the calculated cropped output height – 32px -> 16px each top and bottom. The starting coordinates are ***x*** = 32 and ***y*** = 16.

I hope i could show you how useful and powerful cropping is and you enjoyed reading. Try it and have fun...

command line crop cropdetect encoding FFmpeg ffplay filter

René Calles

... is an experienced video encoding and streaming engineer, passionated developer and also writer and maintainer of this blog.

Not published

Website

Comment

Post It

5 Replies

5 Comments

0 Tweets

0 Facebook

0 Pingbacks

last reply was 1 month ago

Lu.. 1 month ago

Thanks a lot ! finally works! !!

· reply

Lu.. 6 months ago

Nice tutorial René. Many thanks. Can you also make it auto crop, based on the cropdetect results? I'm aiming for cropping a batch of videos in 1 take.

· reply

R. C. 6 months ago

Thanks Luc, and yes, it is possible to create a script to batch convert your videos. You need to process then every video 2 times … first with cropdetect, then parsing the values and finally using the values with the crop filter.

· reply

L 6 months ago

Thanks Rene, this would be awesome. Can you share an example script?

· reply

I (6 months ago

Luc, i don't have anything ready yet. I will let you know when i have something finished. Meanwhile you could check google

for such scripts.

· reply

Follow Me

Recent Posts:

Get the latest FFmpeg on OS X with help of Homebrew
FFmpeg 2.0 is released
Understanding FFmpeg – Part III: Cropping
Understanding FFmpeg – Part II: Scaling video
How to get the latest FFmpeg binaries for Windows

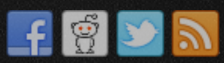
Tags

about me blender blog codecs command line compiling crop cropdetect encoding FFmpeg ffplay filter Homebrew How To Mac OS X MPEG DASH png release resize scale streaming tears of steel video VP8 VP9 webm Windows wordpress x264 XCode

Categories

FFmpeg (9)
General (4)
webm (1)

Follow Me



Legal Notice

Impressum

Archives

July 2013
May 2013
April 2013
March 2013

Tags

about me | blender | blog | codecs | command line | compiling | crop | cropdetect | encoding | FFmpeg | ffplay | filter | Homebrew | How To | Mac OS X | MPEG DASH | png | release | resize | scale | streaming | tears of steel | video | VP8 | VP9 | webm | Windows | wordpress | x264 | XCode

FFmpeg Filtering Guide

FFmpeg has access to many filters and more are added on a regular basis. To see what filters are available with your build see `ffmpeg -filters`.

Contents

Documentation

Refer to the FFmpeg filters documentation for more information and examples for each filter. This wiki page is for user contributed examples and tips, and contributions to this page are encouraged.

Examples

Scaling

Starting with something simple. Resize a 640x480 input to a 320x240 output.

```
ffmpeg -i input -vf scale=iw/2:-1 output
```

`iw` is input width. In this example the input width is 640. 640/2 = 320. The `-1` tells the scale filter to preserve the aspect ratio of the output, so in this example the scale filter will choose a value of 240. See the FFmpeg documentation for additional information.

Speed up your video

See How to speed up / slow down a video for examples.

Filtergraph,Chain,Filter relationship

What follows the `-vf` in an ffmpeg command line is a filtergraph description. This filtergraph may contain a number of chains, each of which may contain a number of filters.

Whilst a full filtergraph description can be complicated, it is possible to simplify it for simpler graphs provided ambiguity is avoided.

Remembering that filters in a chain are separated by commas "," chains by a semicolon ";" and that if an input or output is not specified it is assumed to come from the preceding or sent to the following item in the chain.

The following are equivalent:

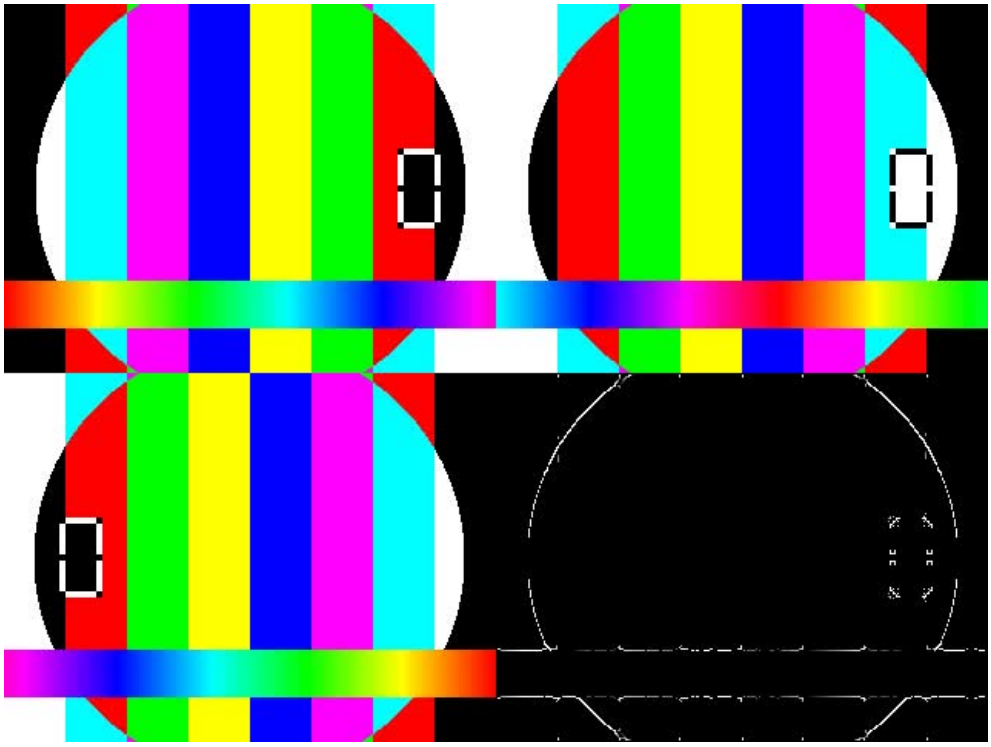
```
ffmpeg -i input -vf [in]scale=iw/2:-1[out] output
ffmpeg -i input -vf scale=iw/2:-1 output                                    # the input and output ar
```

As are:

```
ffmpeg -i input -vf [in]yadif=0:0:0[middle];[middle]scale=iw/2:-1[out] output # 2 chains form, one filt
ffmpeg -i input -vf [in]yadif=0:0:0,scale=iw/2:-1[out] output                 # 1 chain form, with 2 fi
ffmpeg -i input -vf yadif=0:0:0,scale=iw/2:-1  output                         # the input and output ar
```

multiple input overlay in 2x2 grid

Here four inputs are filtered together using the `-filter_complex` option. In this case all of the inputs are `-f lavfi -i testsrc` but could be other inputs. Within the filtergraph the first input is padded to the right and bottom by double its height and the other three inputs are individually filtered using hflip, negate, and edgedetect. The overlay video filter is then used multiple times to arrange of latter three inputs on top of the first one. The offsets used in the overlay filter arrange the inputs into a grid shape.



```
ffmpeg -f lavfi -i testsrc -f lavfi -i testsrc -f lavfi -i testsrc -f lavfi -i testsrc -filter_complex
```

Escaping characters

As described in the documentation, it can be necessary to escape commas "," that need to appear in some arguments, for example the select filter:

```
ffmpeg -i input -vf select='eq(pict_type\,PICT_TYPE_I)' output                    #to select only
```

However an alternative, which also allows for white space within the filtergraph, and which may assist in clarity of reading complex graphs, is to enclose the whole filtergraph within double quotes " " thus:

```
ffmpeg -i input -vf "select=eq(pict_type,PICT_TYPE_I)" output                #to select only I frames
ffmpeg -i input -vf "yadif=0:-1:0, scale=iw/2:-1" output                     # deinterlace then resize
```

Note that the examples given in the documentation mix and match the use of "full quoting" and "\" escaping, and that use of unusual shells may upset escaping. See Notes on filtergraph escaping for more information.

Burnt in Timecode

Using the drawtext video filter.

PAL 25 fps non drop frame:

```
ffmpeg -i in.mp4 -vf "drawtext=fontfile=/usr/share/fonts/truetype/DroidSans.ttf: timecode='09\:57\:00\:
x=(w-tw)/2: y=h-(2*lh): fontcolor=white: box=1: boxcolor=0x00000000@1" -an -y out.mp4
```

NTSC 30 fps drop frame

```
(change the : to a ; before the frame count)_____________________________________________________
                                                                                                  \
ffmpeg -i in.mp4 -vf "drawtext=fontfile=/usr/share/fonts/truetype/DroidSans.ttf: timecode='09\:57\:00\;
x=(w-tw)/2: y=h-(2*lh): fontcolor=white: box=1: boxcolor=0x00000000@1" -an -y out.mp4
```

Scripting your command line parameters

If building complex filtergraphs the command line can get very messy so it can help to break things down into manageable pieces. However one needs to be careful when joining them all together to avoid issues due to your shell and escaped characters.

The following example shows a sample bash script containing a filtergraph of one chain with three filters; yadif, scale and drawtext.

```
#!/bin/bash
# ffmpeg test script

path="/path/to/file/"

in_file="in.mp4"
out_file="out.mp4"
```

```
cd $path

filter="-vf \"yadif=0:-1:0, scale=400:226, drawtext=fontfile=/usr/share/fonts/truetype/DroidSans.ttf: \
text='tod- %X':x=(w-text_w)/2:y=H-60 :fontcolor=white :box=1:boxcolor=0x00000000@1\""
codec="-vcodec libx264  -pix_fmt yuv420p -b:v 700k -r 25 -maxrate 700k -bufsize 5097k"

command_line="ffmpeg -i $in_file $filter $codec -an $out_file"

echo $command_line
eval $command_line
exit
```

Note that the double quotes " around the whole filtergraph have been escaped \" and the filtergraph spans more than one line, the echo command shows the full command as it is executed. Useful for debugging.

The `eval` invocation of the $command_line variable is required to avoid loss of the embedded escaped quotes which occurs if it is absent. Other shells may behave differently.

List of Filters

Filters bundled with libavfilter as of 3.82.100 (as configured with `--enable-gpl`). Filters relying on external libraries, such as frei0r, are not listed here. See FFmpeg filters documentation for more information and examples for each filter.

```
  T. = Timeline support
  .S = Slice threading
  A = Audio input/output
  V = Video input/output
  N = Dynamic number and/or type of input/output
  | = Source or sink filter
.. aconvert         A->A       Convert the input audio to sample_fmt:channel_layout.
.. aecho            A->A       Add echoing to the audio.
T. afade            A->A       Fade in/out input audio.
.. aformat          A->A       Convert the input audio to one of the specified formats.
.. ainterleave      N->A       Temporally interleave audio inputs.
.. allpass          A->A       Apply a two-pole all-pass filter.
.. amerge           N->A       Merge two or more audio streams into a single multi-channel stream.
.. amix             N->A       Audio mixing.
.. anull            A->A       Pass the source unchanged to the output.
T. apad             A->A       Pad audio with silence.
.. aperms           A->A       Set permissions for the output audio frame.
.. aphaser          A->A       Add a phasing effect to the audio.
.. aresample        A->A       Resample audio data.
.. aselect          A->N       Select audio frames to pass in output.
.. asendcmd         A->A       Send commands to filters.
.. asetnsamples     A->A       Set the number of samples for each output audio frames.
.. asetpts          A->A       Set PTS for the output audio frame.
.. asetrate         A->A       Change the sample rate without altering the data.
.. asettb           A->A       Set timebase for the audio output link.
.. ashowinfo        A->A       Show textual information for each audio frame.
.. asplit           A->N       Pass on the audio input to N audio outputs.
.. astats           A->A       Show time domain statistics about audio frames.
.. astreamsync      AA->AA     Copy two streams of audio data in a configurable order.
.. atempo           A->A       Adjust audio tempo.
.. atrim            A->A       Pick one continuous section from the input, drop the rest.
.. bandpass         A->A       Apply a two-pole Butterworth band-pass filter.
.. bandreject       A->A       Apply a two-pole Butterworth band-reject filter.
.. bass             A->A       Boost or cut lower frequencies.
.. biquad           A->A       Apply a biquad IIR filter with the given coefficients.
.. channelmap       A->A       Remap audio channels.
.. channelsplit     A->N       Split audio into per-channel streams.
.. compand          A->A       Compress or expand audio dynamic range.
.. earwax           A->A       Widen the stereo image.
.. ebur128          A->N       EBU R128 scanner.
.. equalizer        A->A       Apply two-pole peaking equalization (EQ) filter.
.. highpass         A->A       Apply a high-pass filter with 3dB point frequency.
.. join             N->A       Join multiple audio streams into multi-channel output.
.. lowpass          A->A       Apply a low-pass filter with 3dB point frequency.
.. pan              A->A       Remix channels with coefficients (panning).
.. silencedetect    A->A       Detect silence.
.. treble           A->A       Boost or cut upper frequencies.
T. volume           A->A       Change input volume.
.. volumedetect     A->A       Detect audio volume.
.. aevalsrc         |->A       Generate an audio signal generated by an expression.
.. anullsrc         |->A       Null audio source, return empty audio frames.
.. sine             |->A       Generate sine wave audio signal.
.. anullsink        A->|       Do absolutely nothing with the input audio.
.. alphaextract     V->N       Extract an alpha channel as a grayscale image component.
.. alphamerge       VV->V      Copy the luma value of the second input into the alpha channel of the f
T. bbox             V->V       Compute bounding box for each frame.
.. blackdetect      V->V       Detect video intervals that are (almost) black.
.. blackframe       V->V       Detect frames that are (almost) black.
TS blend            VV->V      Blend two video frames into each other.
T. boxblur          V->V       Blur the input.
T. colorbalance     V->V       Adjust the color balance.
```

```
T. colorchannelmixer V->V       Adjust colors by mixing color channels.
T. colormatrix      V->V       Convert color matrix.
.. copy             V->V       Copy the input video unchanged to the output.
.. crop             V->V       Crop the input video to width:height:x:y.
T. cropdetect       V->V       Auto-detect crop size.
T. curves           V->V       Adjust components curves.
T. dctdnoiz         V->V       Denoise frames using 2D DCT.
.. decimate         N->V       Decimate frames (post field matching filter).
T. delogo           V->V       Remove logo from input video.
.. deshake          V->V       Stabilize shaky video.
T. drawbox          V->V       Draw a colored box on the input video.
T. drawgrid         V->V       Draw a colored grid on the input video.
T. edgedetect       V->V       Detect and draw edge.
.. extractplanes    V->N       Extract planes as grayscale frames.
.S fade             V->V       Fade in/out input video.
.. field            V->V       Extract a field from the input video.
.. fieldmatch       N->V       Field matching for inverse telecine.
.. fieldorder       V->V       Set the field order.
.. format           V->V       Convert the input video to one of the specified pixel formats.
.. fps              V->V       Force constant framerate.
T. framestep        V->V       Select one frame every N frames.
T. geq              V->V       Apply generic equation to each pixel.
T. gradfun          V->V       Debands video quickly using gradients.
T. haldclut         VV->V      Adjust colors using a Hald CLUT.
.. hflip            V->V       Horizontally flip the input video.
T. histeq           V->V       Apply global color histogram equalization.
.. histogram        V->V       Compute and draw a histogram.
T. hqdn3d           V->V       Apply a High Quality 3D Denoiser.
T. hue              V->V       Adjust the hue and saturation of the input video.
.. idet             V->V       Interlace detect Filter.
T. il               V->V       Deinterleave or interleave fields.
.. interlace        V->V       Convert progressive video into interlaced.
.. interleave       N->V       Temporally interleave video inputs.
.. kerndeint        V->V       Apply kernel deinterlacing to the input.
T. lut3d            V->V       Adjust colors using a 3D LUT.
T. lut              V->V       Compute and apply a lookup table to the RGB/YUV input video.
T. lutrgb           V->V       Compute and apply a lookup table to the RGB input video.
T. lutyuv           V->V       Compute and apply a lookup table to the YUV input video.
.. mcdeint          V->V       Apply motion compensating deinterlacing.
.. mp               V->V       Apply a libmpcodecs filter to the input video.
.. mpdecimate       V->V       Remove near-duplicate frames.
T. negate           V->V       Negate input video.
.. noformat         V->V       Force libavfilter not to use any of the specified pixel formats for the
TS noise            V->V       Add noise.
.. null             V->V       Pass the source unchanged to the output.
T. overlay          VV->V      Overlay a video source on top of the input.
T. owdenoise        V->V       Denoise using wavelets.
.. pad              V->V       Pad input image to width:height[:x:y[:color]] (default x and y: 0, defa
.. perms            V->V       Set permissions for the output video frame.
T. perspective      V->V       Correct the perspective of video.
.. pixdesctest      V->V       Test pixel format definitions.
T. pp               V->V       Filter video using libpostproc.
.. psnr             VV->V      Calculate the PSNR between two video streams.
T. removelogo       V->V       Remove a TV logo based on a mask image.
T. rotate           V->V       Rotate the input image.
T. sab              V->V       Apply shape adaptive blur.
.. scale            V->V       Scale the input video to width:height size and/or convert the image for
.. select           V->N       Select video frames to pass in output.
.. sendcmd          V->V       Send commands to filters.
.. separatefields   V->V       Split input video frames into fields.
.. setdar           V->V       Set the frame display aspect ratio.
.. setfield         V->V       Force field for the output video frame.
.. setpts           V->V       Set PTS for the output video frame.
.. setsar           V->V       Set the pixel sample aspect ratio.
.. settb            V->V       Set timebase for the video output link.
.. showinfo         V->V       Show textual information for each video frame.
T. smartblur        V->V       Blur the input video without impacting the outlines.
.. split            V->N       Pass on the input to N video outputs.
T. spp              V->V       Apply a simple post processing filter.
.. stereo3d         V->V       Convert video stereoscopic 3D view.
.. super2xsai       V->V       Scale the input by 2x using the Super2xSaI pixel art algorithm.
.. swapuv           V->V       Swap U and V components.
.. telecine         V->V       Apply a telecine pattern.
.. thumbnail        V->V       Select the most representative frame in a given sequence of consecutive
.. tile             V->V       Tile several successive frames together.
.. tinterlace       V->V       Perform temporal field interlacing.
.. transpose        V->V       Transpose input video.
.. trim             V->V       Pick one continuous section from the input, drop the rest.
T. unsharp          V->V       Sharpen or blur the input video.
.. vflip            V->V       Flip the input video vertically.
T. vignette         V->V       Make or reverse a vignette effect.
TS yadif            V->V       Deinterlace the input image.
.. cellauto         |->V       Create pattern generated by an elementary cellular automaton.
.. color            |->V       Provide an uniformly colored input.
.. haldclutsrc      |->V       Provide an identity Hald CLUT.
```

```
.. life             |->V       Create life.
.. mandelbrot       |->V       Render a Mandelbrot fractal.
.. mptestsrc        |->V       Generate various test pattern.
.. nullsrc          |->V       Null video source, return unprocessed video frames.
.. rgbtestsrc       |->V       Generate RGB test pattern.
.. smptebars        |->V       Generate SMPTE color bars.
.. smptehdbars      |->V       Generate SMPTE HD color bars.
.. testsrc          |->V       Generate test pattern.
.. nullsink         V->|       Do absolutely nothing with the input video.
.. avectorscope     A->V       Display audio vector scope.
.. concat           N->N       Concatenate audio and video streams.
.. showspectrum     A->V       Convert input audio to a spectrum video output.
.. showwaves        A->V       Convert input audio to a video output.
.. amovie           |->N       Read audio from a movie source.
.. movie            |->N       Read from a movie source.
.. ffbuffersink     V->|       Buffer video frames, and make them available to the end of the filter g
.. ffabuffersink    A->|       Buffer audio frames, and make them available to the end of the filter g
.. abuffer          |->A       Buffer audio frames, and make them accessible to the filterchain.
.. buffer           |->V       Buffer video frames, and make them accessible to the filterchain.
.. abuffersink      A->|       Buffer audio frames, and make them available to the end of the filter g
.. buffersink       V->|       Buffer video frames, and make them available to the end of the filter g
.. afifo            A->A       Buffer input frames and send them when they are requested.
.. fifo             V->V       Buffer input images and send them when they are requested.
```

Test Source

The testsrc filter is basically an auto generated image FFmpeg can inject. For instance

```
ffmpeg -f lavfi -i testsrc=duration=10:size=1280x720:rate=30 out.mpg
```

Would generate a 10 second clip (at 30 fps, so 300 frames total) at 1280x720 (which would then be transcoded and end up in out.mpg file).

Here is ffplay with the same:

```
ffplay -f lavfi -i "testsrc=duration=10:size=1280x720:rate=30"
```

And smptebars example, related:

```
ffmpeg -f lavfi -i "smptebars=duration=5:size=1280x720:rate=30" smptebars.mp4
```

Other Filter Examples

- Fancy Filtering Example – Examples for various psychedelic effects and other weird filtering

Developing your own Filters

- See FFmpeg filter HOWTO

Last modified 11 days ago

Attachments (1)

- multiple_input_overlay.jpg (22.3 KB) - added by *llogan* 15 months ago.

Download all attachments as: .zip

Download in other formats:
Plain Text

« Compare RED, Canon 5D and Lumix GH1 in 1 spot

HDSLR Cinema tool: frame rate conversion »

Convert between framerates (24 fps, 25 fps, 30 fps, ...)

Published on March 17, 2010 in Workflow. 5 Comments
Tags: ffmpeg, opensource.

The 5D originally only shot in 30 fps (NTSC-inspired, undoubtedly, since it was meant for USA news agencies anyway). So when the footage was to be used for TV (PAL in Europe = 25 fps) or for cinema (24 fps), it had to be converted. Also, when you shoot in 720p60 (at 60 fps) so that you can make a nice slow motion effect, you have to 'convert' your material. We already referred to an article about how to 'conform with Cinema Tools/FinalCutPro'. But let's talk some more about this conversion and how to do it with a free open-source tool like `ffmpeg`.

1. Conversion with fixed # of frames.

This is what is referred to as '*conforming*'. You keep the same number of frames, so a 20 second long 30 fps clip (20 x 30 = 600 frames) becomes 25 seconds at 24 fps. The same number of frames are played at a slower speed. The only thing you have to do for the video is to change the metadata of your clip. The audio will have to be played slower too, so will sound lower. If you want to fix this, you have to pitch this up (with audacity or sox).

How to do this with ffmpeg? Well, in two steps:

- extract the frames as rawvideo
  `ffmpeg -i input.mov -f rawvideo -b 50000000 -pix_fmt yuv420p -vcodec rawvideo -s 1920x1080 -y temp.raw`
- recreate the video with new framerate
  `ffmpeg -f rawvideo -b 50000000 -pix_fmt yuv420p -r 24 -s 1920x1080 -i temp.raw -y output.mov`
- To save on needed intermediate disk space, you could send the output of the first to stdout and pipe it to the input of the second

2. Conversion with fixed running length.

This method keeps the total length, but in order to do so, it has to *interpolate* (estimate frames that are between 2 original frames) . In other words, for each second you get 30 input frames and you need to create e.g. 24 output frames. Output frame 3 of that second will be created out of input frames 3 and 4, combined in some way. If this interpolation is done with a low quality/fast algorithm, any steady smooth movement in the input movie, might become jerky and unnatural. Therefor this procedure will be slow (might be slower than real time – you'll need more than 1 hour of conversion time for each hour of footage) The audio will remain untouched, since the total length does not change.

How to do this with ffmpeg? Easy:

- convert framerate in one step:
  `ffmpeg -i input.mov -sameq -r 24 -y output.mov`

Unfortunately, this is quite slow AND low quality. So you might want to look at tools like FinalCutPro to do this kind of conversion. Also, MVtools (AVIsynth) seems to be able to do this better, I still have to check that out.

Recent Posts
Nikon D800 vs Canon 5D Mk III
Dimensions: feature film shot on 5D
Digital cinema aspect ratios
Depth-of-field calculator: now with visualisation!
Composition calculator

HDSLR News
Black Magic Cinema Camera (BMCC)
Beautiful & Befuddling/Canon C100 leads to murky future for mid- to upper-range...
FilmmakerIQ.com
Samsung Threatens to Strand Indian Bloggers in Germany
NoFilmSchool
HDSLR Cinema News

Categories
Select Category

Ads

Archives
Select Month

Information
About
HDSLR Cameras
Tools

Movies
Beyond the Still
DSLR Cinema
DSLR Flix

News
Cinema 5D
EOS HD
NoFilmSchool
Planet5D

People
Byron Shah
Chase Jarvis
Dan Chung
F-Stop Academy
Kevin Shahinian
Oliver Peters
Philip Bloom
Rodney Charters
Shane Hurlbut
Stu Maschwitz
Tom Guilmette
Vincent Laforet

Tools
HDSLR Tools

5 Responses to "Convert between framerates (24 fps, 25 fps, 30 fps, …)"

TheToad
May 1, 2010 at 10:30 pm

Unfortunately the way how ffmpeg (also the ffmpeg broadcast version http://code.google.com/p/ffmbc/) read the h264 mov file is not very good: you get the highlight very compressed and you lose information if you want do a good grading!!!
Personally, after a long research, i use another way :
Using avisynth with QTinput plugin (to open mov file) and open the avs file with virtualdub or ffmpeg to export in your favorite format and frame rate.

if you use open avs file with ffmpeg you have to add flipvertical() command in avs file; that is not userfull if you use virtualdub(i don't know why)
I will post soon some where a small tutorial to how menage all this and also export in loss-less free codec (huffyuv or Lagarith)
sorry for my poor English
i hope this is user full

yellow
August 25, 2010 at 11:53 pm

@TheToad, I find QTInput scales the luma, even with latest QT Alternative installed. It appears to scale the luma twice though, so levels appear correct until you check them on a waveform or histogram to see combing and data loss.

So i use Haali Media splitter and remux into a matroska container, then use AVISynth & Haali bundled .dlls to read the mkv's to decode the h264.

Also for anyone reading, never use FFmpeg to convert to RGB always use AVISynth. FFmpegs swscale expands the luma crushing blacks and clipping whites, and on default settings for interpolation looses about 50% of colour data compared to AVISynth's conversion.

This all works on Linux too with Wine.

Ome Two
December 28, 2010 at 3:56 pm

I tried to "upframe" – convert mp4 with 23.976 to 30 frames rate. Does not work. I mean, output file still has 23.976 frames rate.

alienresident
March 19, 2012 at 11:12 pm

Found this post and it was very helpful start. I wanted to give back some additional info I discover for anyone who comes here via google.

1. This is how you pipe the output of step 1 into step 2 so you don't have to create (& then delete) the temp.raw file

ffmpeg -i input.mov -f rawvideo -b 50000000 -pix_fmt yuv420p -vcodec rawvideo -s 1920×1080 -y pipe:1 | ffmpeg -f rawvideo -b 50000000 -pix_fmt yuv420p -r 24 -s 1920×1080 -i – output.mov

Secondly, by default this will create a h264 file if you want to use a different codec such as Apple Pro Res (only in the newer builds of FFmpeg)

ffmpeg -i input.mov -f rawvideo -b 50000000 -pix_fmt yuv420p -vcodec rawvideo -s 1920×1080 -y pipe:1 | ffmpeg -f rawvideo -b 50000000 -pix_fmt yuv420p -r 24 -s 1920×1080 -i – -vcodec prores -profile 2 output.mov

serg_ulixes
June 8, 2012 at 10:02 am

I think I found a more elegant way to do that using "-vsync passthrough" option. It is

conforming my 59.97fps footage to slow motion 23.98 without frame skipping:

for %%a in ("*.mov") do ffmpeg -i "%%a" -s 1280×720 -sws_flags lanczos -r 23.98 -vsync passthrough -vcodec dnxhd -b 110M -mbd rd -acodec copy "%%~na.DNxHD.mov"

« Compare RED, Canon 5D and Lumix GH1 in 1 spot

HDSLR Cinema tool: frame rate conversion »

HDSLR Cinema tool: frame rate conversion « HDSLR Cinema News
Pingback on Mar 29th, 2010 at 2:11 pm

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Submit

« Compare RED, Canon 5D and Lumix GH1 in 1 spot

HDSLR Cinema tool: frame rate conversion »