x264 Video Encoding Frontend

By Alvin Lam

California Polytechnic State University

Computer Science Department

June 1, 2015

Abstract

x264 is a free video codec for encoding video streams into the H.264/MPEG-4 AVC format. It has become the new standard for video encoding, providing higher quality with a higher compression than that of XviD.[1] x264 provides a command line interface as well as an API and is used in popular applications such as HandBrake[2] and FFmpeg[3]. Advanced Audio Coding (AAC) is a very popular audio coding standard for lossy digital audio compression. AAC provides a higher sound quality than MP3 at similar bitrates.[4] This senior project describes the design and implementation of a x264 video encoding frontend that uses these codecs to encode videos. The frontend provides a simple and easy-to-use graphical user interface. Subtitles are preserved across encodes and the resulting encoded file is stored in a Matroska container format.

Acknowledgments

I would like to thank my project adviser Dr. Gene Fisher for his support and guidance throughout the entire project. I would also like to thank Gaurav Singh (fOOt) for letting me join his encoding team and introducing me to encoding in the first place. Lastly I would like to thank Vörös Szebasztián (durzegon), Mike Andries (Maiku), and the rest of the team for being such great software testers and for demanding even more features.

Table of Contents

List of Figures

1 Introduction

I am a huge fan of Japanese animated series (anime) many of which are not localized or translated in the United States. There are many fansubbers who translate, sub, encode, and upload these animated series for fans to download. In June 2014, I was lucky enough to join an encoding team that focuses on re-encoding fansubbed anime into 10-bit with AAC audio, providing a smaller file size with little affect on quality. As re-encoders, sometimes we encode an entire series at once or want to import multiple audio and subtitle streams.

1.1 Problem Description

Our encoding team uses a script to do our encoding; this requires us to keep multiple copies of the script for different encodes. Scripts provide very limited functionality, currently they only encode a single episode at a time that must be renamed and moved to the script directory. The scripts also fail to support exporting source audio (if the source encode already uses AAC audio) and multi-audio sources. We have to modify the script to change the parameters or whenever we encode a different series. Although there are many x264 encoding frontends available (HandBrake[2], FFmpeg[3], and MeGUI[5]). However, most of these tools are extremely cluttered and difficult to use. All of these tools also use the official x264 build which lacks several preferred settings. As the more experienced programmer on the team, I decided to create a more lightweight and user-friendly interface than currently exist.

1.2 Project Overview

In this paper, *encoding* is the process of translating a source to a digital format for playback on media devices. *Muxing* is the process of merging multiple different streams together. One may want to encode a recorded video to allow playback on certain devices. Others may want to *re-encode* a video to further compress and reduce the source file size and to mux it with additional audio or subtitle streams.

My goal was to create a lightweight and user-friendly Graphical User Interface (GUI) for encoders to use. The application will allow users to queue up multiple files to encode and save/load profiles. There are tool-tips for every setting, providing a friendly introduction to new encoders. More advanced users may also overwrite or add additional commands through a custom command line input. This will allow users to easily encode/re-encode videos into the H.264 format without relying on a script method of encoding or trying to understand what each setting does. Details regarding the tools and encoding process are explained below.

1.3 Encoding Process

A variety of tools must be used to encode a video. A common process used by most encoders is a shell script or batch script that runs the codecs in order with the parameters via command line. This script method is preferred over most frontend applications since it provides the most control over encoding settings.

There are many different builds of x264 that provides additional settings. For this application, I will be using x264 rev2431 tMod[6]. x264 is used for the video encoding. FFmpeg[3] is used to demux the audio from the source and NeroAACEnc[7;8] is then used to encode the demuxed audio. Finally mkvmerge[9] is used to mux the subtitles with the encoded video and audio into a Matroska media container. Matroska is becoming the standard of multimedia container formats, allowing many audio, video, and subtitle streams to be stored in a single file.[10]

1.4 Outline

Further sections of the report describe important aspects of my project. Section 2 describes the technical specs of my application. Section 3 introduces my finished application along with screen shots of the interface and how a user would interact with it. Section 4 follows up with a review the GUI design and implementation including issues I experienced. Section 5 shows how I test the application and obtained user feedback. Section 6 details my conclusion and future work. Lastly, Appendix A contains documentation of important x264 encoding settings with their respective command line parameters.

2 Technical Specifications

2.1 Language and Framework

I chose to use Python[11] since it is a very high level language and Python code is much shorter than most alternatives, allowing me to quickly push out a working prototype of my GUI.

I used PyQt, a Python binding of the GUI toolkit QT C++ framework.[12] Qt is a very powerful and intuitive GUI toolkit compared to the default Python GUI frameworks and is widely used industrially.

2.2 Dependencies

The following are publicly available resources used in the project:

QDarkStyleSheet[13] – A dark themed stylesheet for QT

MediaInfo[14] – Displays metadata of files

x264 rev2431 tMod[6] – A custom build a x264, providing additional preferred settings and algorithms

FFmpeg[3] – A multimedia framework with the power to decode, encode, transcode, mux, demux, stream, and filter audio

NeroAAC[8] – AAC Audio encoder

MKVToolNix[9] – Set of tools to create, inspect, and alter Matroska files

3 User Guide

3.1 Main Window

When a user first starts the application, they will see the main window as shown in Figure 1. Here users can add/remove video files, view its media information, change profiles, set the output directory, and open the options or settings window.

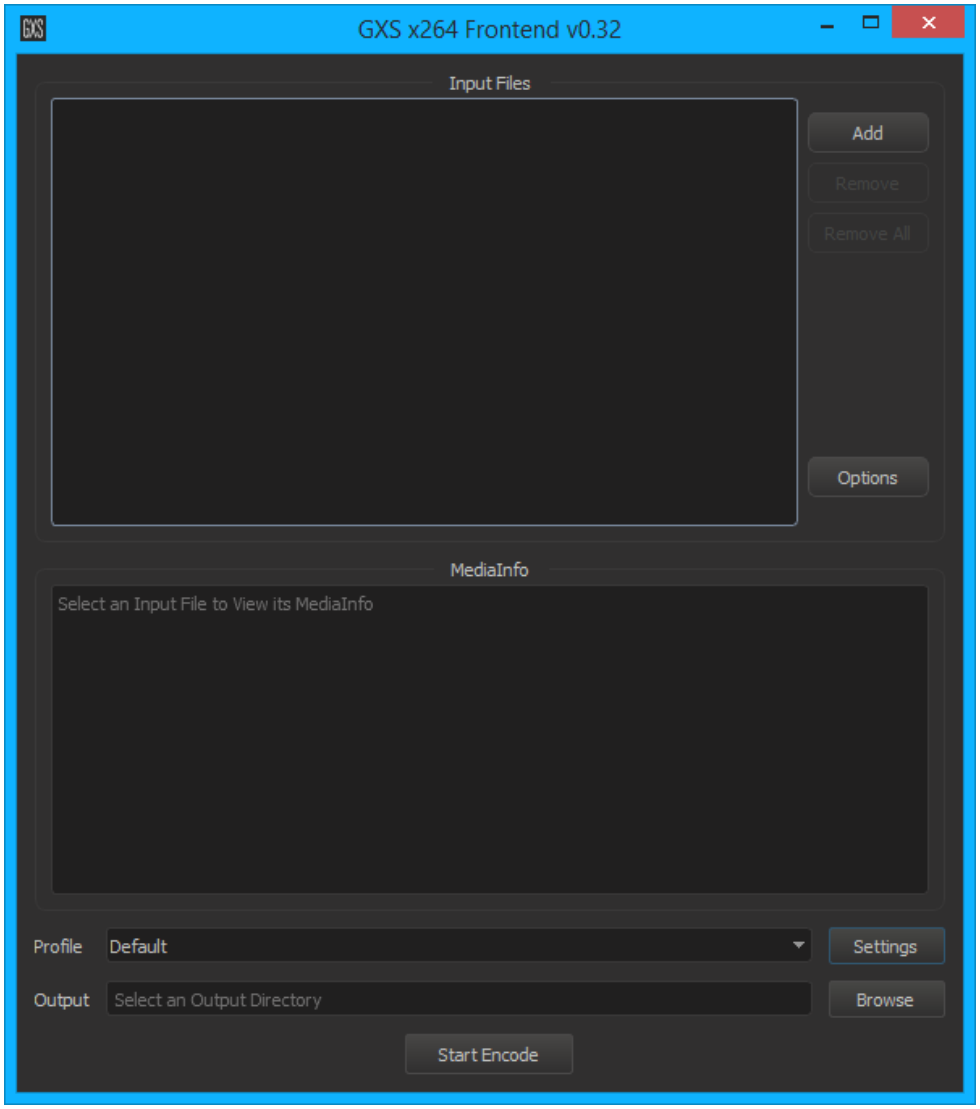

Figure 1: Main User Interface

3.2 Managing Video Files

Users can add video files to encode by dragging the files over as seen in Figure 2.

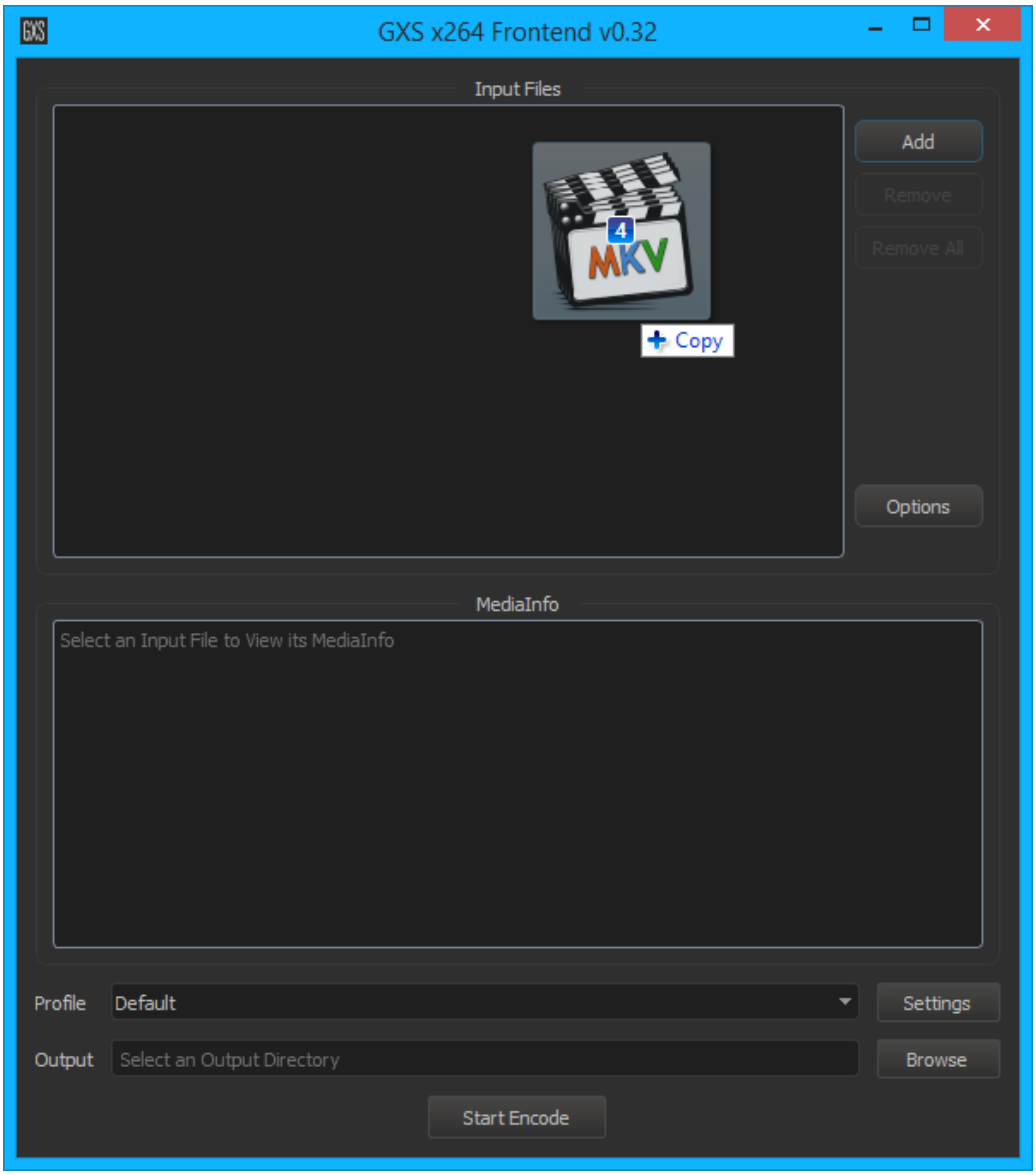

Figure 2: Dragging Videos to Encode

Users can also add video files by clicking the 'Add' button and browse for the video(s) as shown in Figure 3. As many videos can be added as desired.

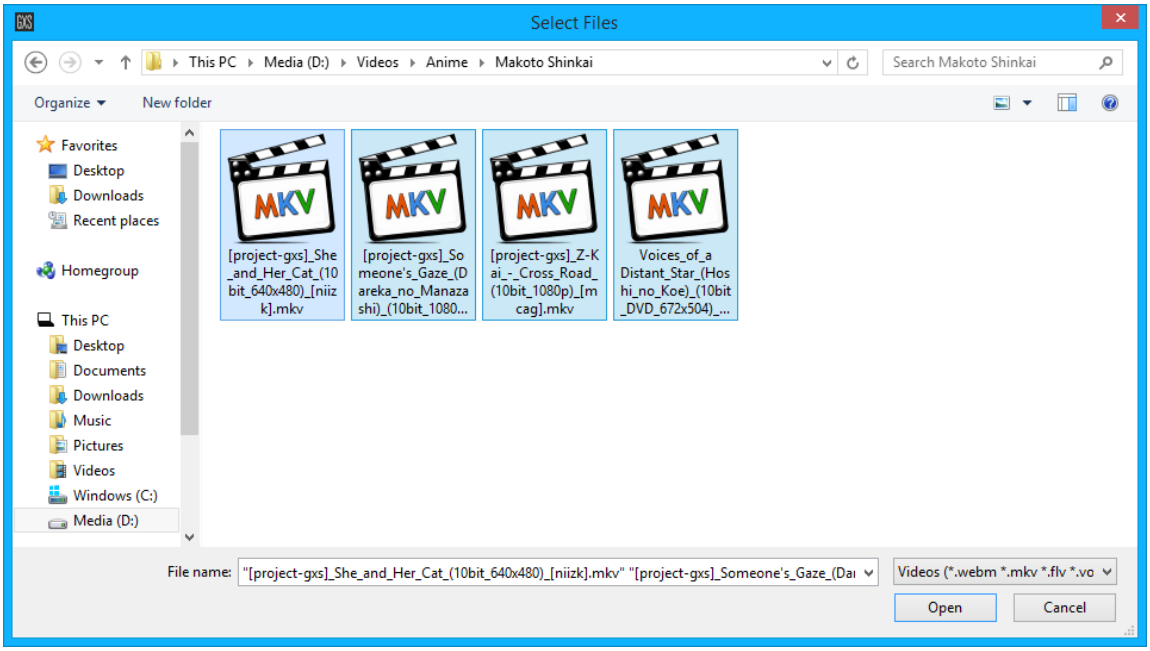

Figure 3: Browsing for Video(s) to Add

The videos added will appear in the 'Input Files' list by their filename. Users can select any video to view its MediaInfo metadata as seen in Figure 4.

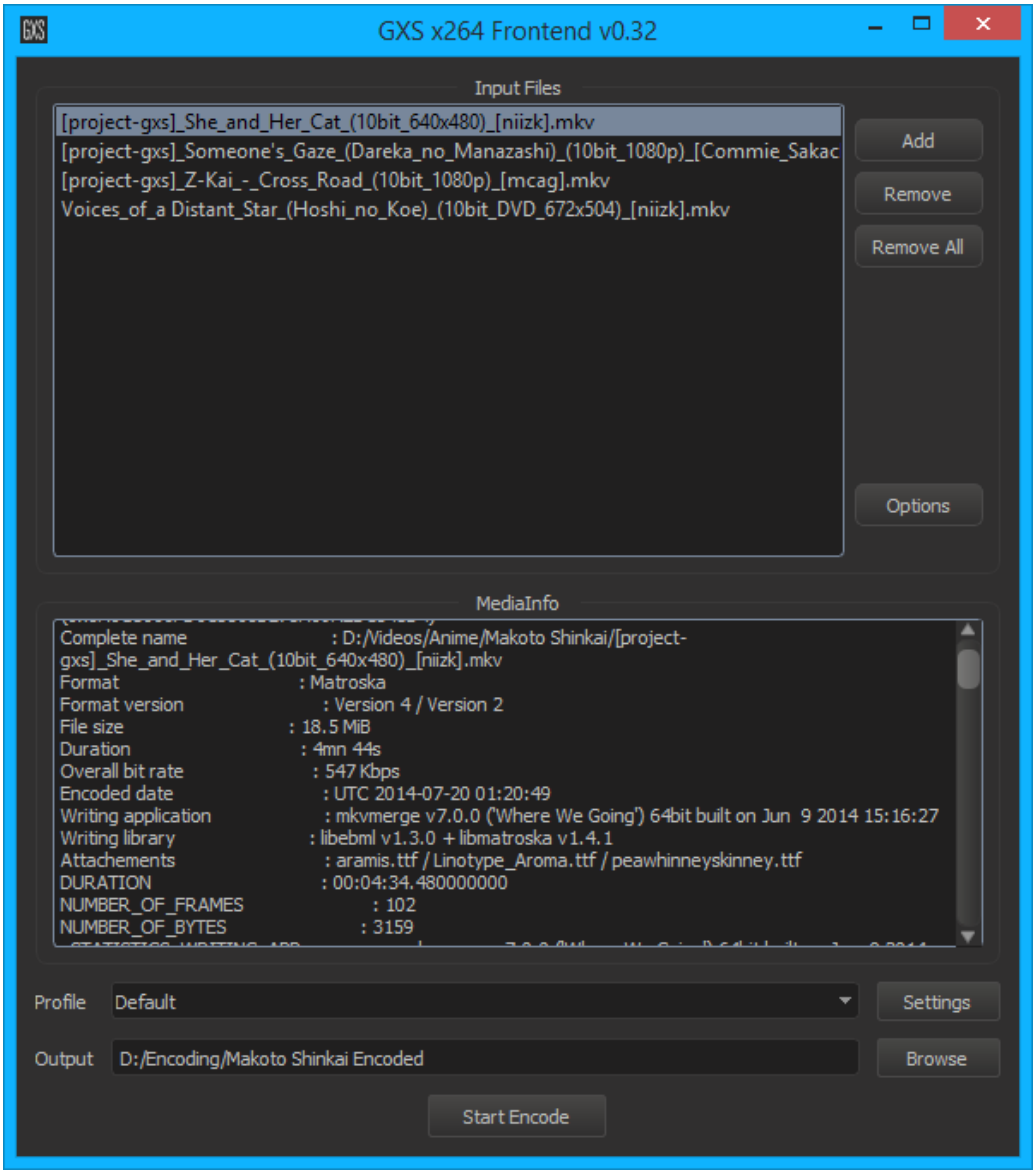

Figure 4: Viewing File's MediaInfo

Users can click and drag videos in list to reorder the encoding queue as shown in Figure 5. Multiple files can be selected and moved/removed this way.

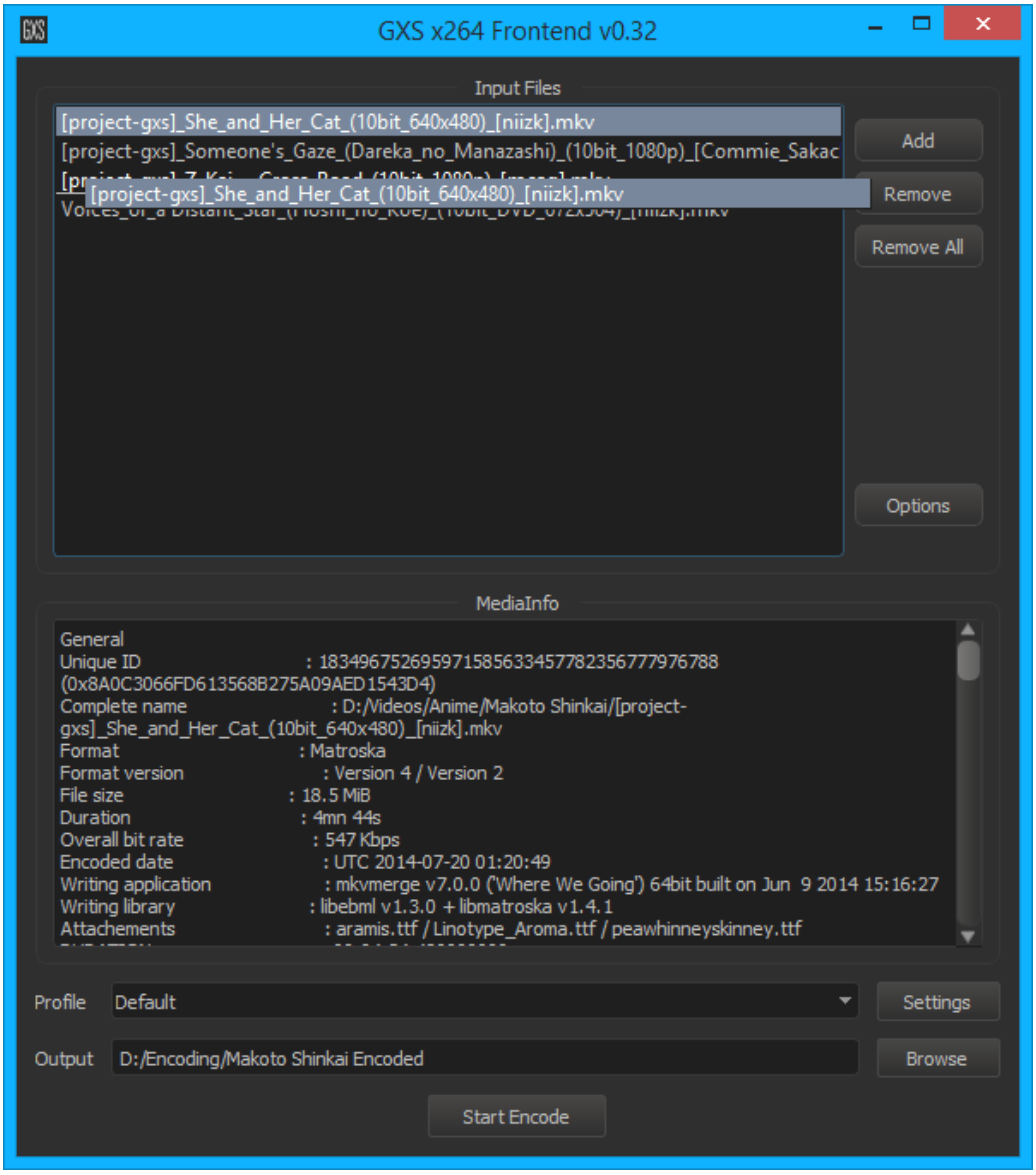

Figure 5: Reordering Encoding Queue

3.3 System Options

There are several options available to the user as shown in Figure 6. Two options can be checked to remember the last used profile and output directory. The first option can be enabled to shutdown the computer after all encodes are completed. If the user accidentally had this setting enabled, the application will warn the user during the encode process and the user can also abort the shutdown following completion.

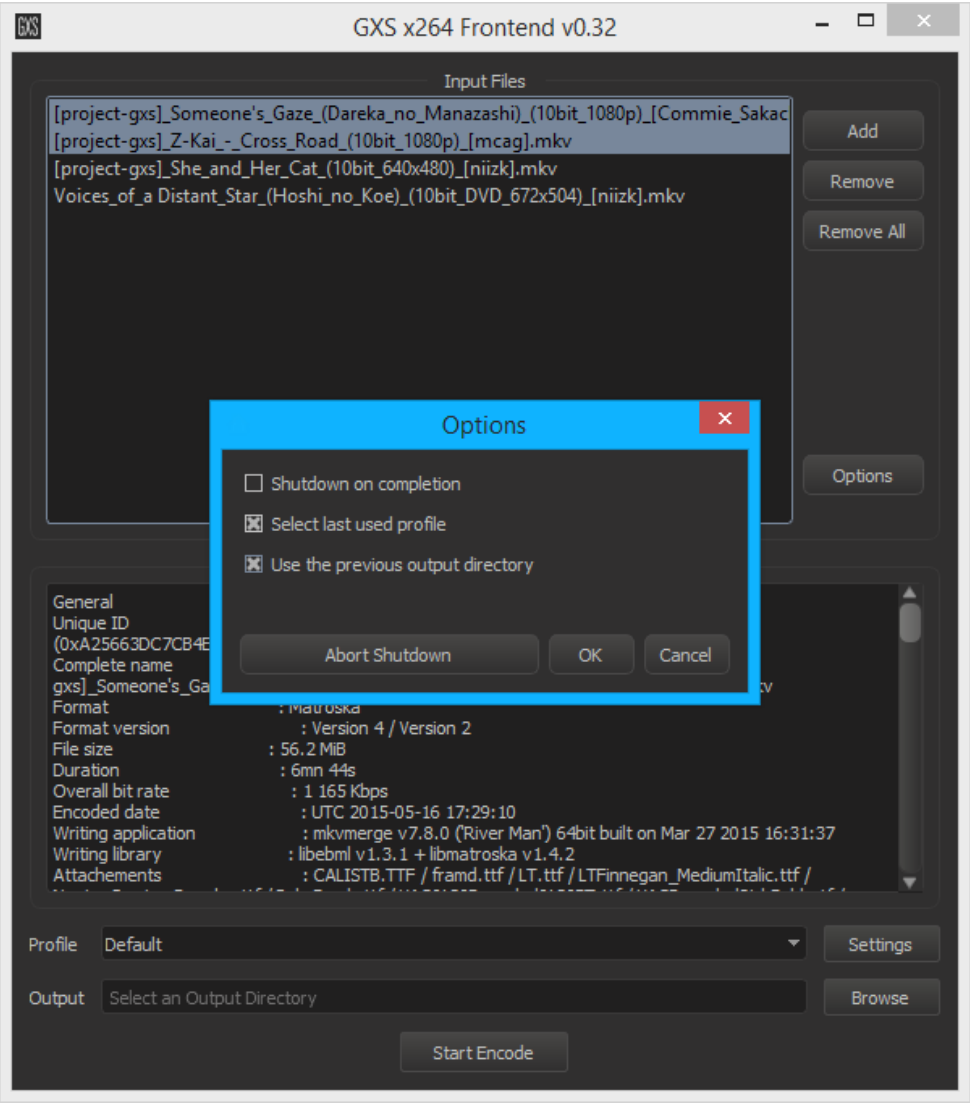

Figure 6: System Options

3.4 x264 Settings and Profiles

When users click on the 'Settings' button on the main window, the 'Encoding Settings' window will be shown as seen in Figure 7. If there are no previous profiles, a 'Default' Profile will be created.

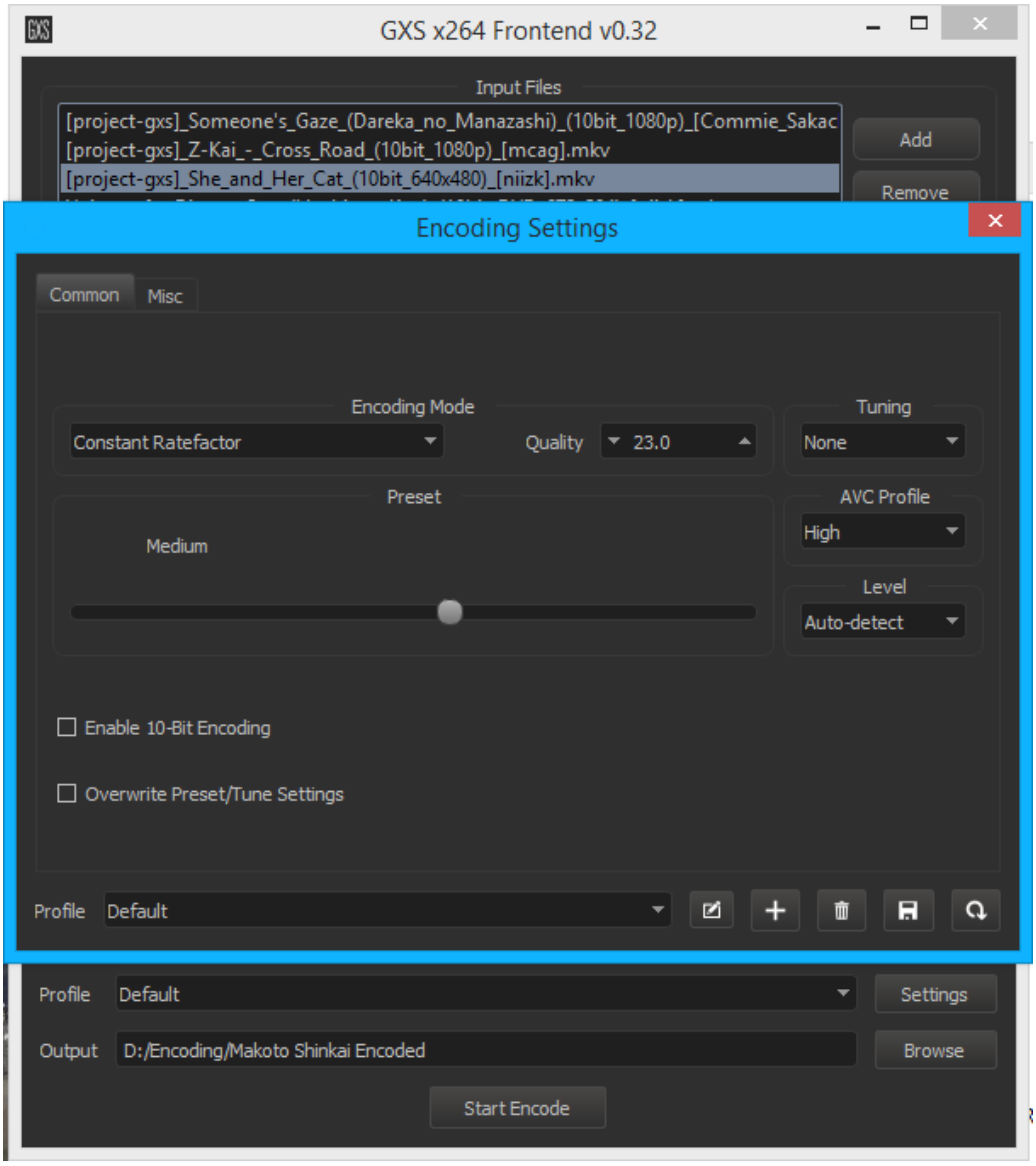

Figure 7: x264 Settings Window

Located on the bottom of the settings window are profile management buttons. Users can rename, add, delete, save profiles, and restore the last saved profile settings. Adding a profile is shown in Figure 8. Users can also use a drop-down menu to select and load any existing profiles.

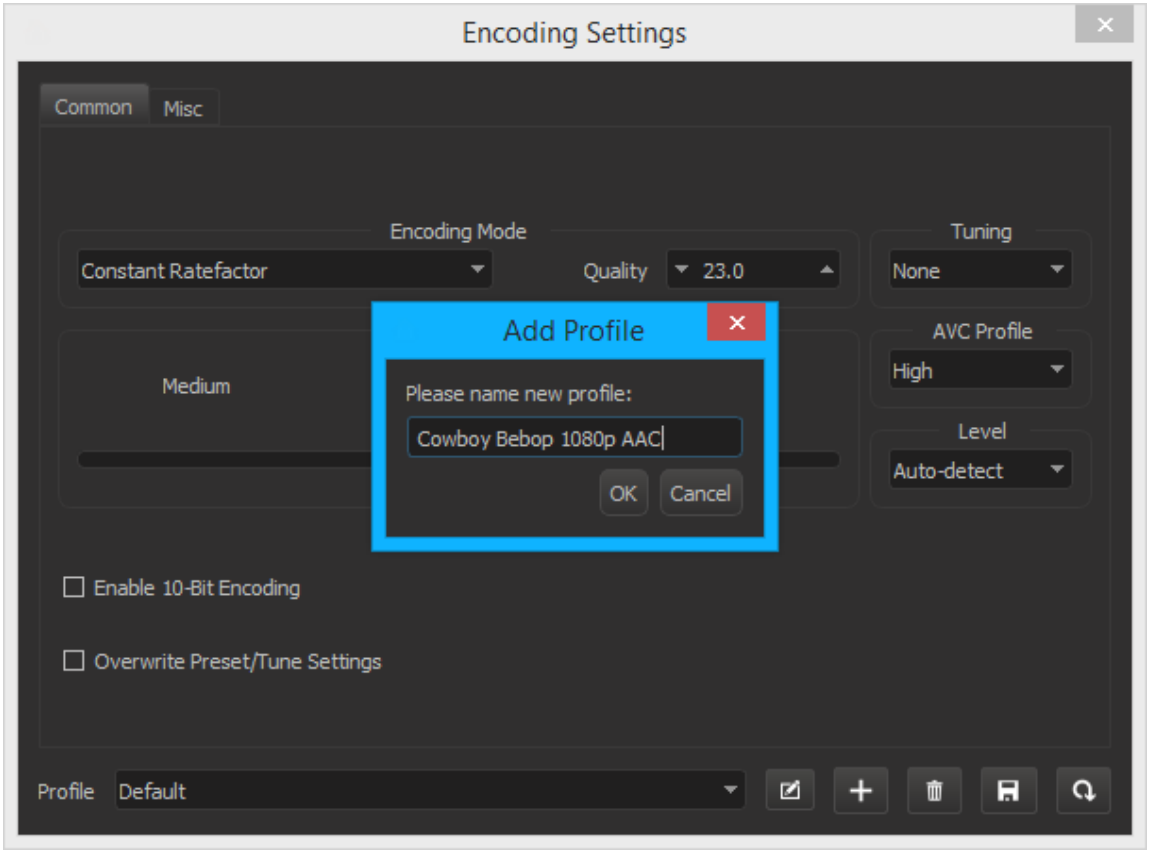

Figure 8: Adding a New Profile

This newly created profile had its settings changed to showcase the different kinds of settings available. As seen in Figure 9, There are tooltips for each setting explaining the functions of the setting and its commandline output. When the 'Overwrite Preset/Tune Settings' is checked, three additional settings tabs will become available.

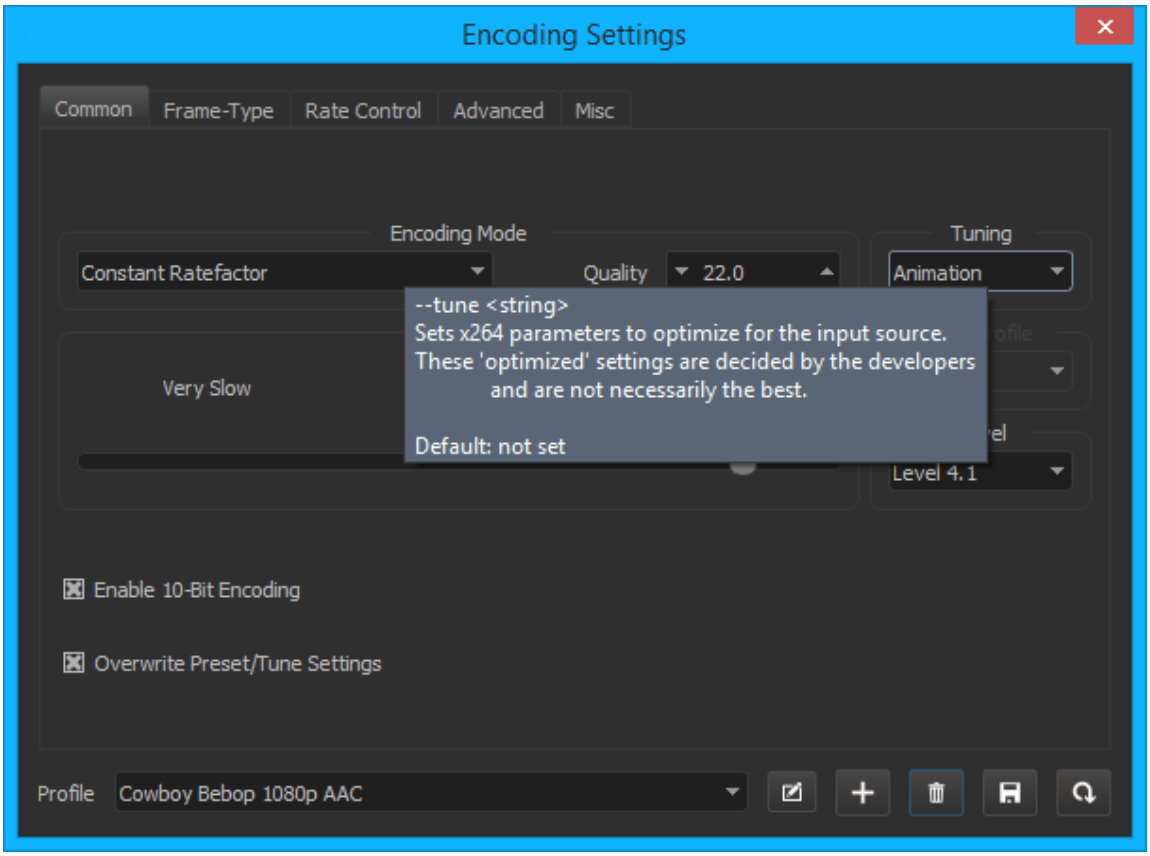

Figure 9: Tooltips and Additional Tabs

The 'Frame-Type' settings tab is shown in Figure 10.

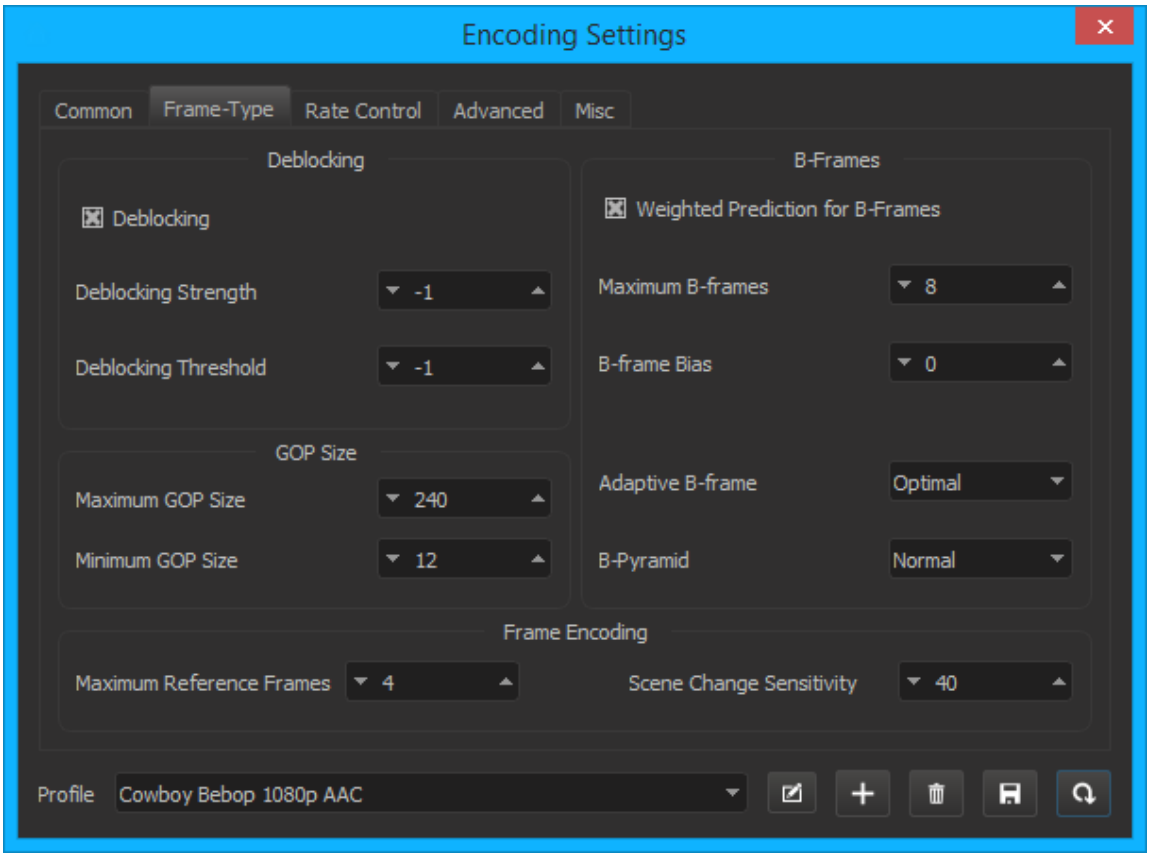

Figure 10: Frame-type Settings

Several settings are dependent or influenced by other settings as seen in Figure 11 when deblocking is disabled. An example of a drop-down menu is shown as well.

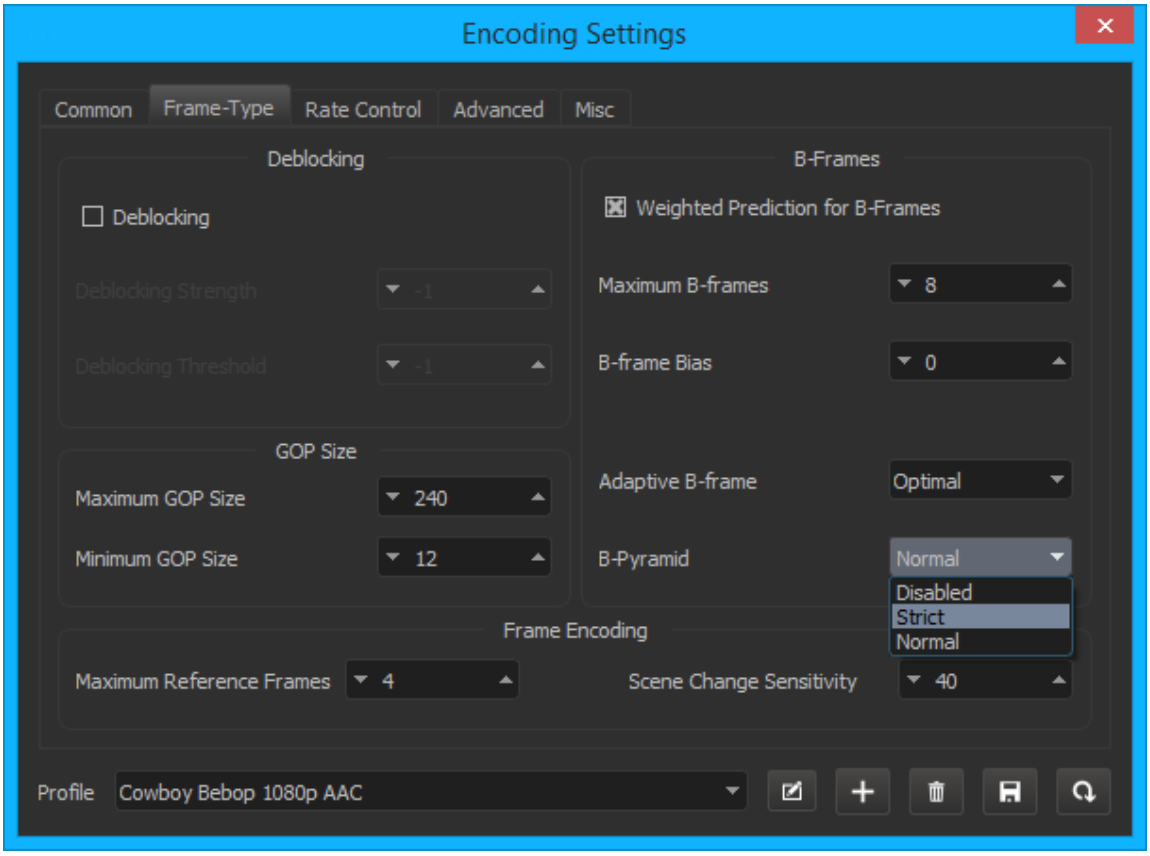

Figure 11: Setting Dependencies and Drop-down Menu

The 'Rate Control' settings tab is shown in Figure 12.

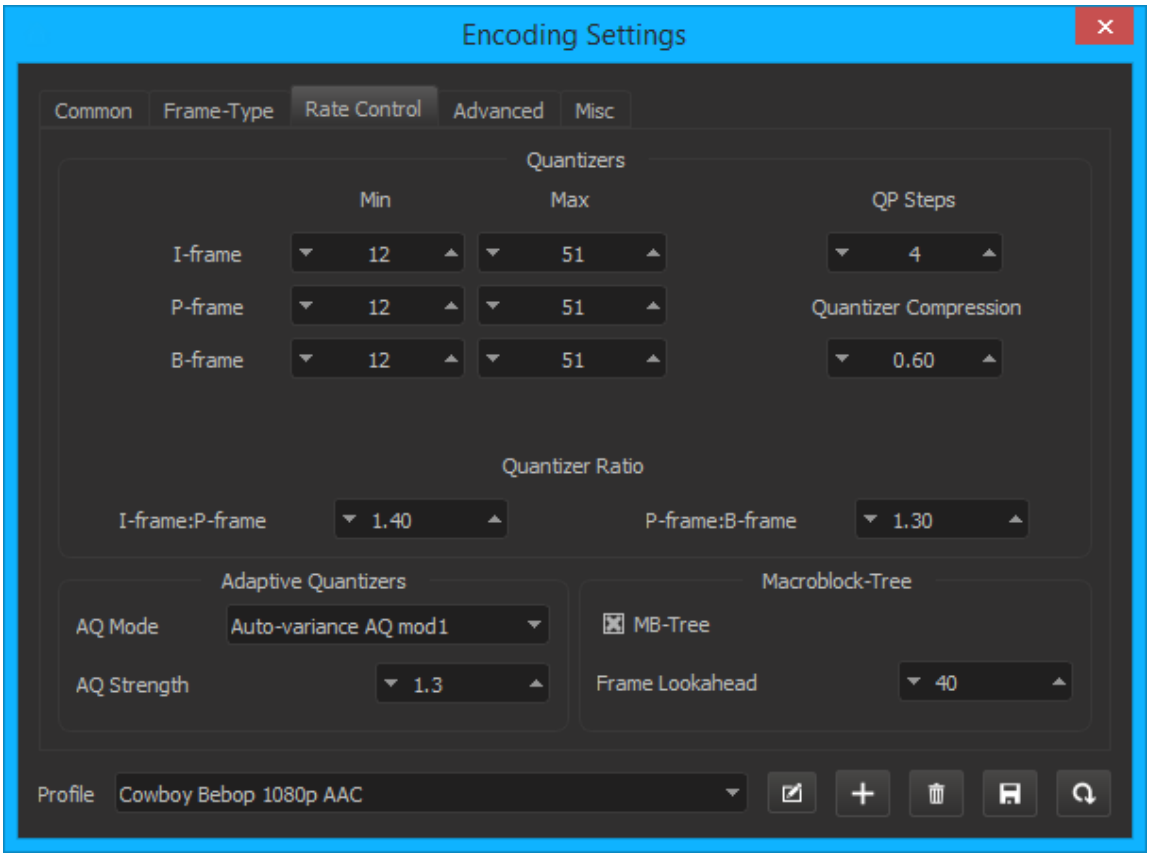

Figure 12: Rate Control Settings

The 'Advanced' settings tab is shown in Figure 13.

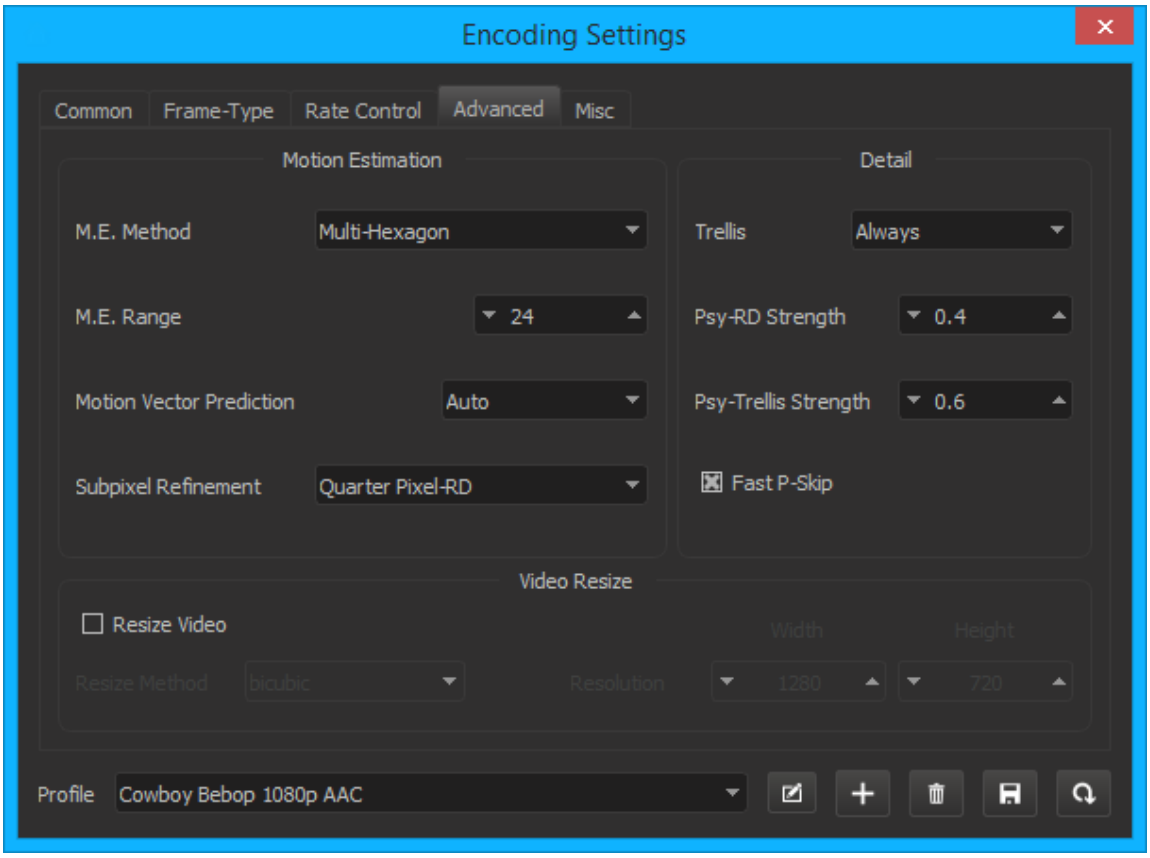

Figure 13: Advanced Settings

The 'Misc' settings tab is shown in Figure 14. We find the audio options here. Users can choose to use the source audio or select the AAC audio quality to use.

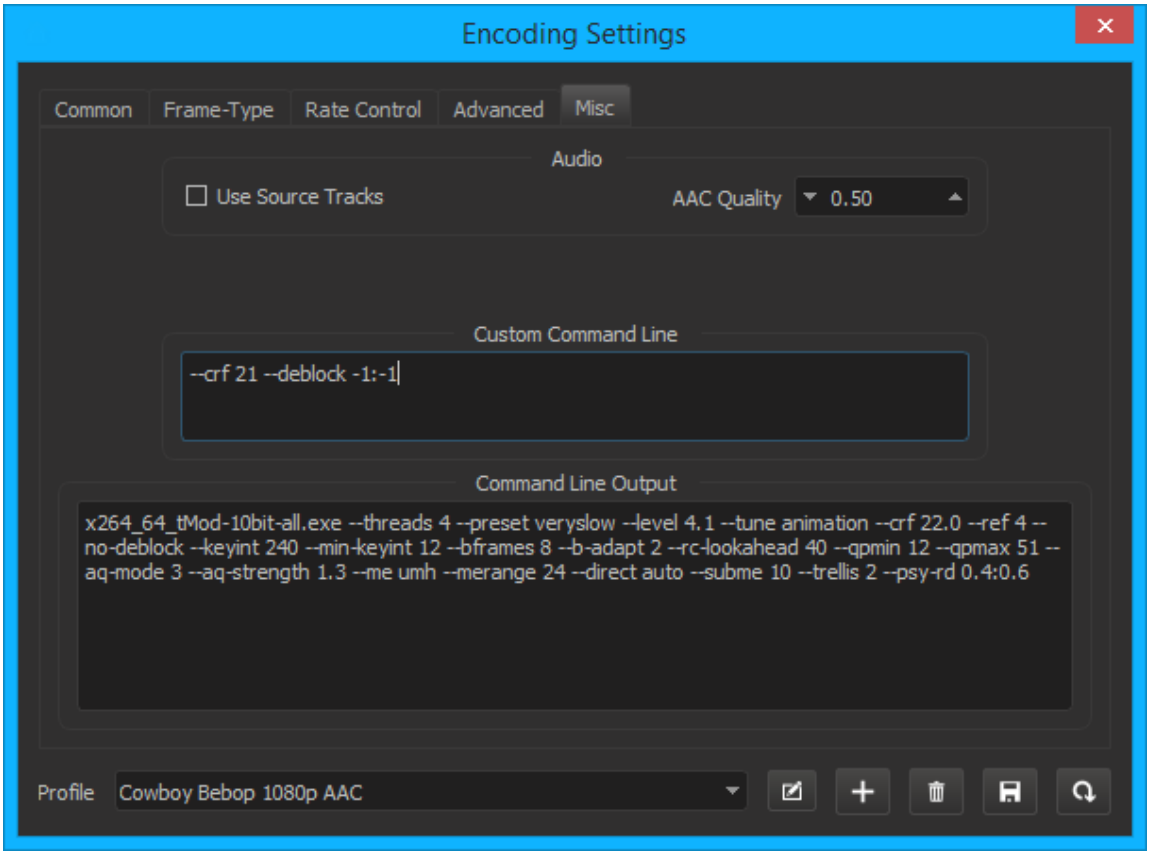

Figure 14: Misc Settings

As seen in the figure, the 'Command Line Output' is shown for the current settings. Advanced users can overwrite existing settings with their own or add additional settings in the 'Custom Command Line'.

3.5 Encoding Process

Once users have created a profile, they can select their desired profile and start the encode. During the encoding process, an 'Output Log' is shown in place of the MediaInfo as seen in Figure 15, which keeps track of the encoding process.

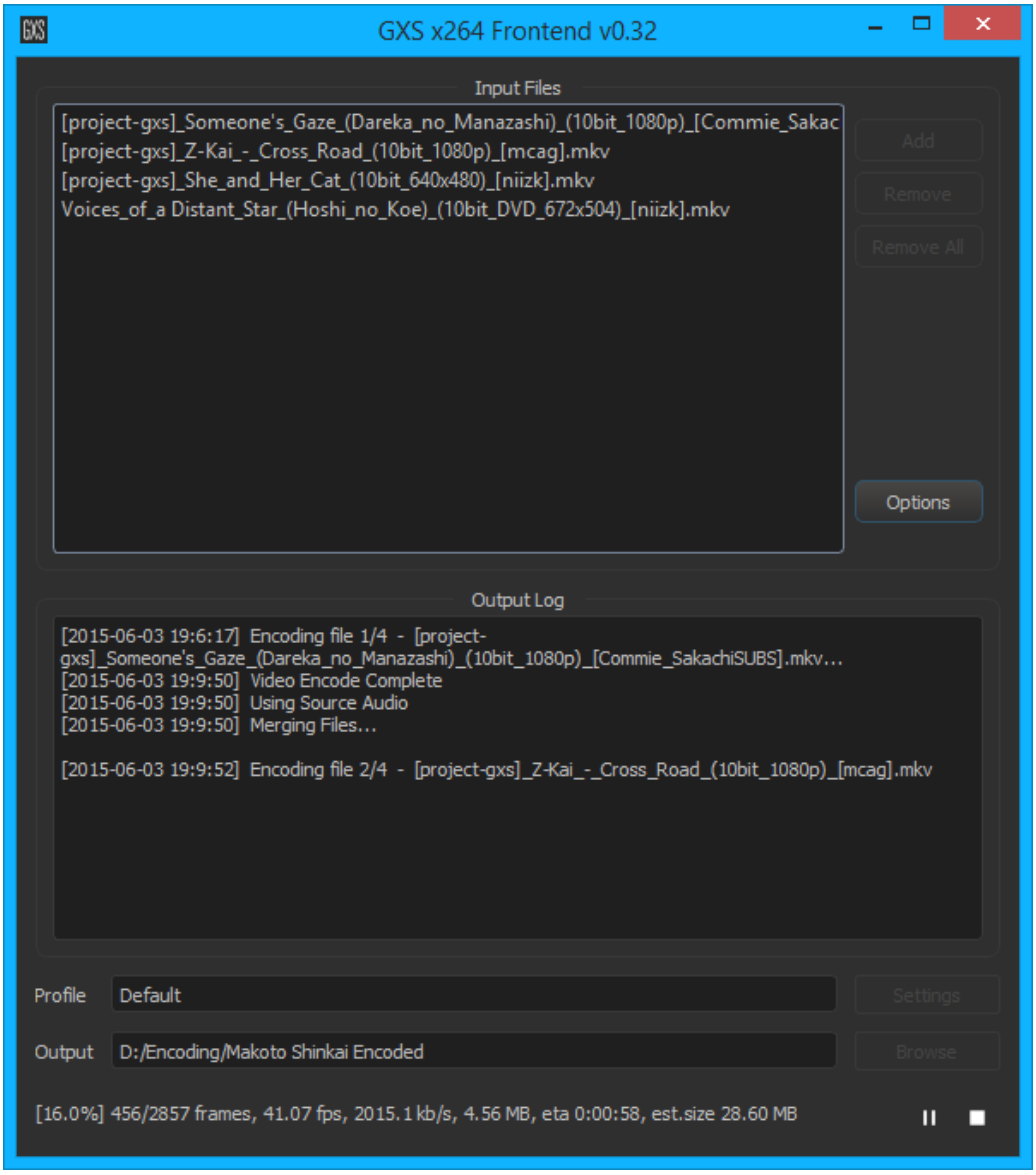

Figure 15: Encoding Process

The current progress is shown at the bottom of the interface (current percentage encoded, current frames encoded/total frames, current frames per second (fps), encoding bit rate, estimated time, and estimated file size). The current encode can be paused/resumed or stopped at any time.

4 GUI Design and Implementation

There were several issues I encountered during the design and implementation process.

4.1 Profiles and Settings

In an attempt to create a light-weight and simple user interface, I wanted to avoid adding an abundance of unnecessary features and settings. I decided to only implement the x264 settings that I deemed the most important but gave users the ability to input additional parameters via command line.

A desired feature was support for creating profiles to save certain settings. These profiles must persist across separate separate executions. I solved this by saving the settings in a configuration file saved under the profile's name. When launching the application, the configurations files are detected and loaded.

x264 has many parameter dependencies, some settings will affect others or require a certain setting to be enabled. For example, if the maximum B-frames is set to 0, then B-frame bias, Adpative B-frame, and B-Pyramid should be disabled since those settings require at least 1 B-frame to function. Or if Subpixel Refinement is set to Quarter Pixel-RD, Trellis must be locked at always on.

There are also the preset options (--preset, --tune) that sets certain x264 parameters unless we overwrite them. Neglecting to overwrite these presets does not break the encoding process but give results that were not desired. To resolve this problem, the system which settings were affected by the preset and check if the user set a different value for those settings so I can tell x264 to overwrite those settings. and give a visual cue in the settings window so the user understands that certain settings are affected by others.

4.2 Edge Cases and User Error

One of the first things I realized that I had to do, was prevent incorrect use of the application. The system does not allow users to encode something that is not a video, so it refuses any files that are not known video formats. The system displays an error dialog when users attempt to start the encoding process without any videos added or when an output directory has not been set. Some users may forget that they set their computer to shutdown when the encoding process is complete, so the system warns the user that shutdown on completion is enabled when a new encode is started as well as give users the ability to abort a shutdown if they are present.

The application also warns if the user attempts to exit when there is an encode in process or stop an encode, all current process will be lost (completed encodes will still exist in the output directory).

Since there are several x264 parameter dependencies, I needed to make sure users understand certain settings affect others. So the system had checks on every dependency and gives visual cues to the user to notify them of those dependencies. This includes disabling and enabling certain settings when a certain parameter is set.

More than one occasions, the software testers were confused by certain settings. One didn't understand how to get the MediaInfo to work. So I added a placeholder text telling the users to 'Select an Input File to View its MediaInfo'.

4.3 Process Management

Once I started linking all my settings to the x264 codec, there was a huge problem. When I started the encode, the application would switch from the main window process to the x264 process. Since the main window process was waiting for the x264 process to complete, the GUI essentially stopped responding. This lead me to look into asynchronous processing that would allow the main window process to continue running while reading and displaying process updates from x264. I used the Qprocess class from qt to accomplish this. QProcess allowed me to start a new process separate from the current process and catch signals from the new process, allowing me to display command line ouput or start the next process once the first completes.

However, one user ran into an issue where the audio encoding process would crash the application. I personally was unable to replicate the bug but understood it stemmed from how I managed processes. So to ensure that processes stop and start as intended, following the completion of each process, I terminate all communication with that process and delete it, reinitializing the process the next time around. I stop and terminate all communication with all running processes before reinitializing them each time: a new encode is started, an encode is stopped or completed, and a process is completed. This was able to resolve the bug.

5 End-User Testing and Validation

As part of an encoding team, I had about four dedicated members who are very willing to use and test my application and provide valuable feedback.

5.1 End-User Testing and Feedback

I emailed everyone on the team notifying them of my project and gave them a brief introduction (features and design choices). I then gave them a link to a shared spread sheet where the team can provide feedback as shown in Figure 16.

The latest version of the application is posted on the spreadsheet. Text marked in red are resolved. An answer or solution is provided for every feedback.

https://mega.co.nz/#!PoUAjDha!i_ZhIxA0XswQO6HLUvqpuHfXVDDB	**Solution**
Bugs	
--preset and --tune overwrites other default settings (RESOLVED)	make settings overwrite --preset and --tune but add an additional checkbox (Use present/tune settings), that way users can just use --preset or --tune for quick settings, and by disabling the "Overwrite preset/tune settings", the user will force the x264 parameters regardless of preset or tune
got Not Responsing 3 times: 1. when i queued 5 files and after the first finished crashed; 2. when i queued 3 files and it was encoding the second files crashed, 3. when i tried a single file encode, before finished with it, it's crashed (i managed to encode 3 files so far) screenshot: http://s23.postimg.org/9otesiocr/crash.png	
2 files encode, crash on analyzing audio stream of second encode. Also works if you stop an encode and restart it (RESOLVED same as above)	Fixed, reinitialize subprocesses on new encodes, next encodes, before encodes, and after each subprocess has accomplished its task
Undertermined language of the source video or audio will cause mkvmerge to fail (RESOLVED)	Handle additional case for undetermined languages
Confusing Layout or Interaction	
What setting gets you subme 10 or aq-mode 3, etc?	I plan to add tooltips for every setting and what cmd-line they correspond to
Mediainfo for files	placeholder text in mediainfo text box "Select an Input File to View its MediaInfo"
Thread count in settings?	Currently decided that if you need your CPU for intensive processing (like gaming), you can simply pause the encode. Anything else we use our computer for, browsing, watching anime, etc does not require more than 10% of our CPU and can be dedicated towards encoding.

Figure 16: Bugs and Confusing Layout/Interactions

I also ask my team what new functionality they would like to see next, along with what I plan to add as seen in Figure 17.

https://mega.co.nz/#!PoUAjDha!i_ZhIxA0XswQO6HLUvqpuHfXVDDB	**Solution**
Suggestions	
To be able to change thread counts. (Or i missed that settings somewhere?)	Will add that setting in, will default to your number of threads but allow users to change it
See Custom Command Line changes in Command Line Output	
To have CRC values automatically added at the end of final encoded file.	
To make media info behave and orient its details in perfect columns.	
Be able to add/remove files in middle of encode	
To be added	
Run batch encode as separate batch script (so everyone that has issues won't have issues, or can at least see the output via cmdline)	
Complete tool tip for every x264 parameter.	
Settings for "shutdown on completion, save last used directory/profile, other things?" (ADDED)	
Pause + Stop button for encoding process (ADDED)	
Custom Command Line functionality (ADDED)	Will not account for invalid input, but catches invalid inputs on encode and warns the user.

Figure 17: Suggestions and To Be Added

Testers would either chat or email me regarding the specifics of their bugs and post screen-shots so that I can replicate it. Certain feedback was very important such as confusing layout and interaction.

5.2 Testing and Validation

To ensure the application is correctly passing the encoding parameters to x264, I would encode a video using certain encoding parameters through the command line. I immediately check to see what x264 parameters are set at the beginning of the encode as seen in Figure 18.

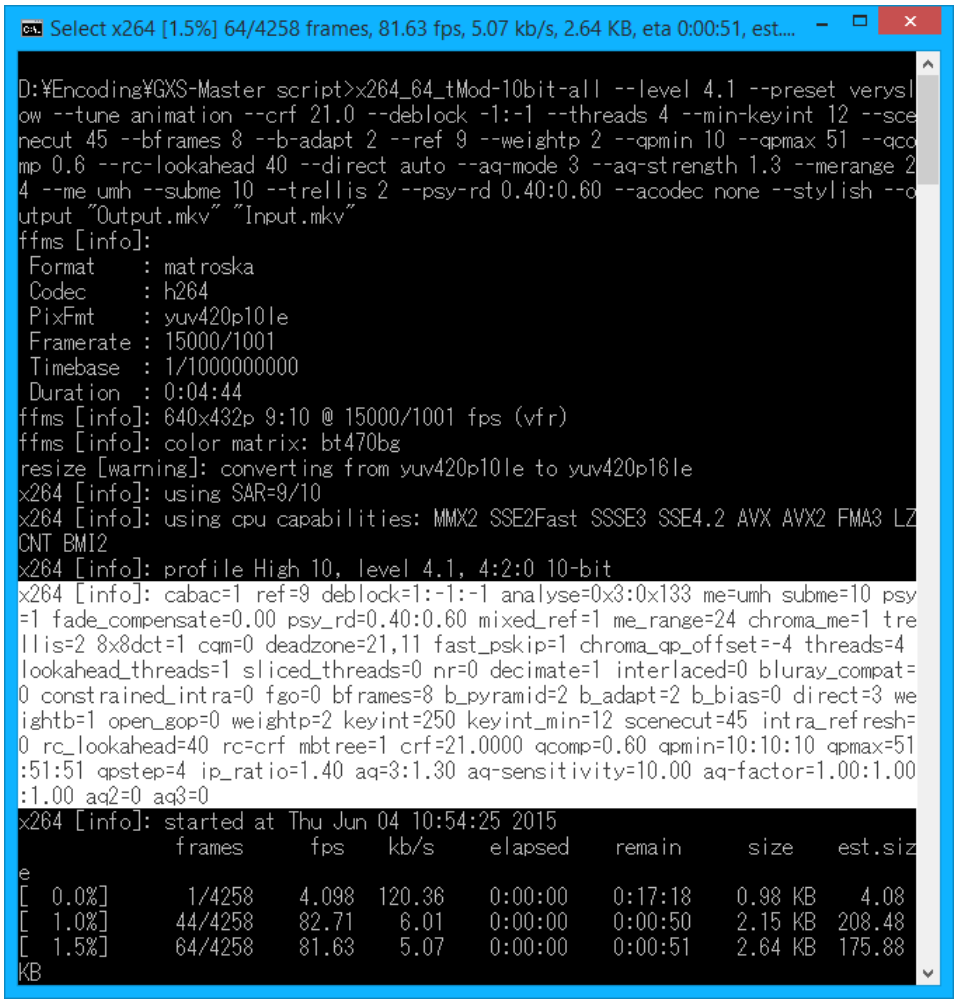

Figure 18: x264 Info Output

Now I use the same settings on my application and make sure the config file is save with the correct parameters. I then start the encode on my application with debug enabled to print the x264 parameters and confirm they have the exact same output.

To automate this process, I can create and modify the configuration files to match the expected x264 parameters as seen in Figure 19.

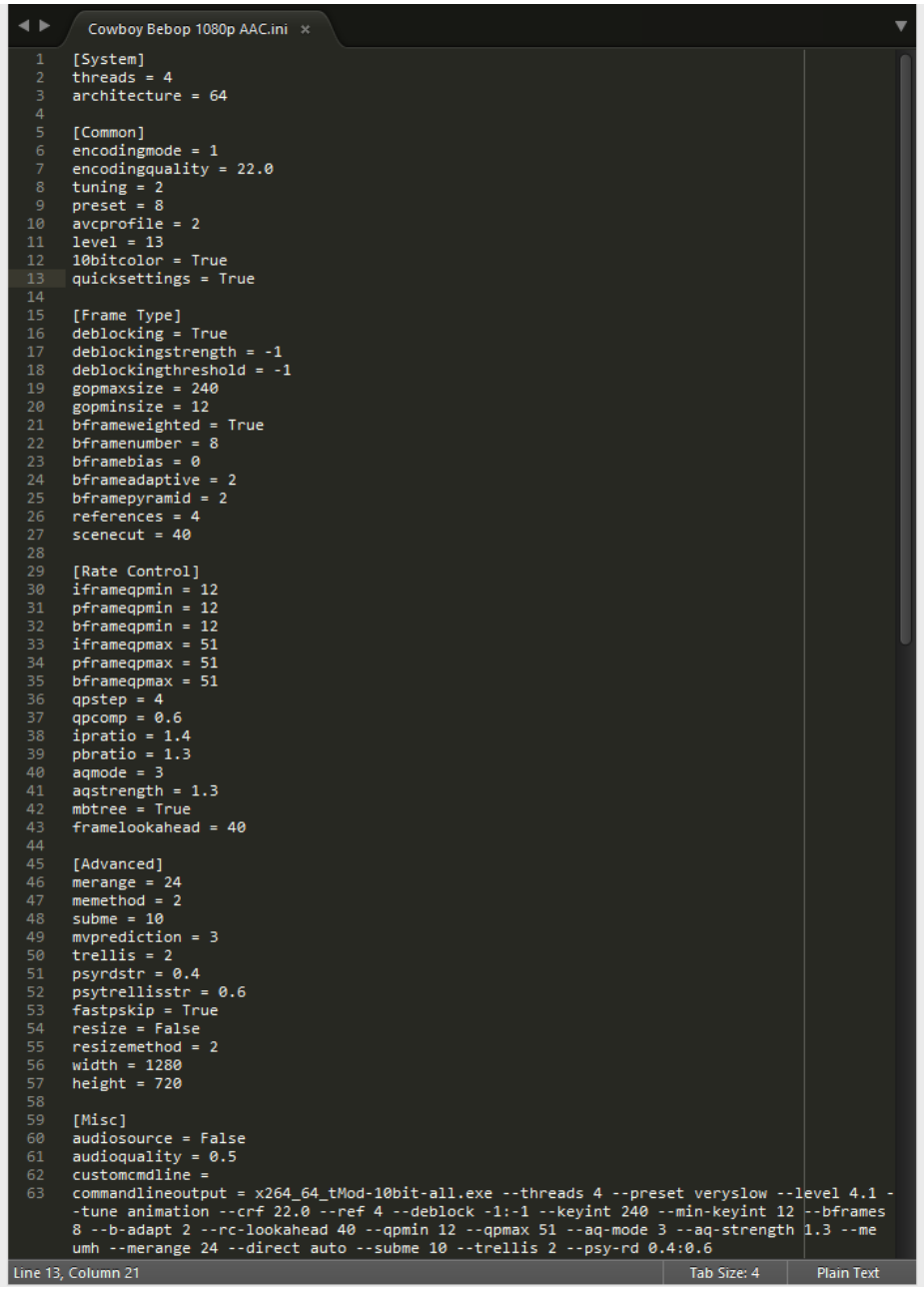

Figure 19: Configuration File

I can import these configuration files to the application and ensure the settings are as expected. I can then start an encode and check the x264 parameters to confirm they have the desired output.

6 Conclusion and Future Work

As someone new to encoding, I learned a great amount of the encoding process, tools, and encoding parameters used. A documentation of the most important x264 parameters is given in Appendix A. I also learned a lot about GUI programming and design. I understand why certain functionality is important especially to new users. Learning the qt library was a plus.

I still plan to add additional functionality including those suggested by my team. I plan to add tooltips explaining each setting. I also plan to update the command line output when the custom command line is updated along with appending a checksum to the end of every file so that every file name is unique (this is a common practice among encoders). As new updates of each tool or codec are released, I will ensure that my application continues to work with them.

A Appendix A – x264 Parameters

The following settings are not exhaustive, only settings deemed to be most useful are included. Some settings are not implemented in the official build of x264. The full list of settings can be found in the help file with the codec.

A.1 Presets

A.1.1 --preset <string>

Sets x264 parameters to balance compression efficiency with encoding speed. Value should generally be set to a slower value. Placebo should be avoided as it gives negligible increase in compression efficiency for a much slower encoding speed.

Values: ultrafast, superfast, veryfast, faster, fast, medium, slow, slower, veryslow, placebo.

Default: medium

Recommended: slow, veryslow

A.1.2 --tune <string>

Sets x264 parameters to optimize for the input source. These 'optimized' settings are set by the developers and are not necessarily the best.

Values:

- film – intended for high-bitrate/high-quality movie content. Lower deblocking is used here.
- animation – intended for animated content, where deblocking is boosted to compensate for larger, flat areas. More reference frames are used.
- grain – this should be used for material that is already grainy. Retains more grains by using a higher aq-strength.
- stillimage – like the name says, it optimizes for still image encoding by lowering the deblocking filter.
- psnr and ssim – these are debugging modes to optimize for good PSNR and SSIM values only.
- fastdecode – disables CABAC and the in-loop deblocking filter to allow for faster decoding on devices with lower computational power.
- zerolatency – optimization for fast encoding and low latency streaming.

Default: not set

A.1.3 --profile <string>

Encoding Profiles representing a sub-set of the encoding parameters to target specific decoders. Enabled for 8-bit only.[15;19]

Values:

- baseline – --no-8x8dct --bframes 0 --no-cabac --cqm flat --weightp 0
- main – --no-8x8dct --cqm flat
- high
- high10 – Support for bit depth 8-10.
- high422 – Support for bit depth 8-10. Support for 4:2:0/4:2:2 chroma subsampling.
- high444 – Suport for bit depth 8-10. Support for 4:2:0/4:2:2/4:4:4 chromas subsampling.

Default: high

A.2 Frame-types

A.2.1 Intraframe (I-frame)

A frame that can stand on its own, requiring no other frames carrying all the information needed to be decoded. I-frames use relatively more space than other frames but are much quicker to decode. These frames create a random access point for playback seeking.[16;17]

A.2.2 Predicted Frame (P-frame)

A interframe that requires information from the preceding frame to decode. P-frames take up less data than I-frames.[16;17]

A.2.3 Bidirectional Predicted Frame (B-frame)

A interframe that requires information from the surrounding I- and P-frames. B-frames take up less data than P-frames but is slower to decode as it requires information from two frames. B-frames improve the compression and quality by a considerable amount but requires a lot of motion estimation (significantly increasing encoding/decoding speed). Another important feature of B-frame is the ability to skip it. A decoder can decide to skip a B-frame if it is behind schedule, which allows fast forwarding to be considerably faster.[16;17]

A.3 Frame-type Options

A.3.1 --keyint <integer>

Sets the max interval between IDR-frames(aka keyframes, Group of pictures (GOP) length). This trades off seeking file seekability with compression efficiency. IDR-frames are 'delimiters' in the stream and are I-frames, meaning players can start decoding from the nearest I-frame instead from the beginning. Therefore they can be used as seek points in a video.[18–20]

When encoding Blu-ray, broadcast, live streaming or other similar videos, a significantly lower GOP length may be required.

Default: 250

Recommended: 250

A.3.2 --min-keyint <integer>

Sets the minimum GOP length, minimum distance between IDR-frames. Playback can only be started at an IDR-frame. Generally the minimum value should be the framerate of the video.[18–20]

Default: auto (min(--keyint/10, --fps))

Recommended: auto

A.3.3 --no-scenecut

Disables adaptive I-frame decision.

Default: not set

A.3.4 --scenecut <integer>

Sets the threshold for I/IDR frame placement (scene change detection). For every frame, x264 calculates a metric to estimate how different it is from the previous frame. If the value calculated is lower than the scenecut value, a 'scene-cut' is detected. An I-frame is placed if it has been less than –min-keyint frames since the last IDR frame, otherwise an IDR frame is placed.[18–20]

Default: 40

Recommended: 40

A.3.5 --no-weightb

Disables weighted prediction for weightb. Enabling weightb produces B-Frames that are more accurate by allowing non-symmetric weighting of reference frames. This feature has a negligible impact on speed. Since weighted B-frames generally improves visual quality, it is better to have it enabled.[18–20]

Default: not set
Recommended: not set

A.3.6 --b-bias <integer>

Controls the likelihood of B-frames being used over P-frames. Values larger than 0 increase the weighting towards B-frame, while values less than 0 do the reverse.[18–20]

Default: 0
Recommended: 0

A.3.7 --bframes <integer>

Sets the maximum number of concurrent B-frames that can be used. The average quality is controlled by --pbratio.[18–20]

Default: 3
Recommended: 16

A.3.8 --b-adapt <integer>

Sets the adaptive B-frame placement algorithm, deciding whether a P- or B-frame is placed.[18–20]

Values:

- 0 – Disable. Always pick B-frames
- 1 – 'Fast' algorithm, speed slightly increases with higher –b-frames setting (use –b-frames 16)
- 2 – 'Optimal' algorithm, slower, speed decreases with higher –b-frames setting.

Default: 1
Recommended: 2

A.3.9 --b-pyramid <value>

Allows B-frames to be used as references for other frames. B-frames references have a quantizer between I-frames and P-frames. This setting is generally beneficial but increases the Decoded Picture Buffer (DPB) size required for playback which may have

compatibility issues. Quality increase is negligible and increases decoding complexity but increases compression.[18–20]

Values:

- none – do not allow B-frames to be used as references.
- strict – allow one B-frame per minigop to be used as reference, Bluray standard.
- normal – allow many B-frames per minigop to be used as reference.

Default: normal

Recommended: normal, strict (if abiding by Blu-ray standard).

A.3.10 --ref <integer>

Controls the size of the DPB. The value controls the number of previous frames each P-frame can use as references. A higher value increases the DPB, which means hardware playback devices usually have strict limits for the number of refs they can handle. If adhering to the H.264 spec, the max refs for 720p and 1080p are 9 and 4 respectively.[18–20]

Default: 3

Recommended: 16 (animation), 9 (720p), 4 (1080p)

A.3.11 --deblock <integer>:<integer>

One of H.264 main features that controls the amount of blocking artifacts from motion estimation with the tradeoff of detail. This requires a small amount of decoding CPU but can significantly increase quality. There are two values used for deblocking, called deblocking strength (alpha deblocking) and deblocking threshold (beta deblocking). The first value is alpha deblocking and the second is beta deblocking.[18–21]

Alpha deblocking affects the total amount of deblocking applied on the picture. A higher alpha value increases the amount of deblocking but destroys more detail.[21]

Beta deblocking determines if something in a block is a detail when deblockingis applied to it. Lower beta values apply less deblocking to more flat blocks when details are present.[21] Deblocking values should not be altered past +/-2.

Default: 0:0

Recommended: -1:-1 (animation)

A.3.12 --no-deblock

Disables deblocking.

Default: not set (deblocking enabled)

Recommended: not set

A.3.13 --no-cabac

CABAC is the default entropy encoder used by x264. Although it is somewhat slower on both the decoding and encoding end, it offers improved compression on sources. CABAC is required for trellis quantization.[18–20]

Default: not set

Recommended: not set

A.4 Rate Control

A.4.1 --qp <integer>

Encodes the video in Constant Quantizer (CQ) mode. The value specifies the P-frame quantizer. The quantizer used for I- and B- frames are derived from –ipratio and –pbratio.[18–20]

Values: 0-81 (0=lossless)

Default: not set

Recommended: --crf

A.4.2 bitrate <integer>

Encodes the video to a set bitrate to achieve a targeted file size but quality level is unknown. The value given is the bitrate in kilobits/sec (1000 bits/sec).[18–20]

Default: not set

Recommended: --pass

A.4.3 --crf <float>

Constant Ratefactor, is a type of rate control in which 'quality' is targeted. The value specifies the quantizer. The idea of CRF is to give the same or higher perceptual quality in a smaller size. This is achieved by using less bitrate for high motion frames and redistribute these bits for still frames. The reason behind this is that high motion frames are deemed less important as a decrease in quality is difficult to notice while details in a still frame are more noticeable and would require a significantly higher bitrate to achieve a good perceptual quality.[18–20;22–24]

CRF takes less time than a 2pass bitrate encode as the first pass is skipped. However it is impossible to predict the bitrate and size of a CRF encode.

Values: -12.0-51.0 (-12=lossless – 18=high – 23=average – 28+=low)

Default: 23.0
Recommended: 17.0-21.0

A.4.4 --rc-lookahead <integer>

This setting sets the number of frames for the encoder to consider as well as other factors before making a decision on the current frame type and quality.[18;19]

Default: 40
Recommended: 40

A.4.5 --qp-min <integer>[:<integer>:<integer>]

Defines the minimum quantizer used. The lower the value, the closer the output is to the source. The first value is for I frames, the second is for P frames, and the third is for B frames.[18–20]

Default: 0:0:0
Recommended: 0:0:0

A.4.6 --qpmax <integer>[:<integer>:<integer>]

Defines the maximum quantizer used. May want to set value lower (30–40) but adjustments are not necessary.[18–20]

Default: 81
Recommended: 30+

A.4.7 --aq-mode <integer>

Adaptive Quantization Mode is used to better distribute the available bits in the video.[18;19;23;25]

Values:

- 0 – Do not use AQ.
- 1 – Variance AQ which allows AQ to redistribute bits across video and within frames.
- 2 – Auto-variance AQ which attempts to adapt strength per frame (experimental – tends to over reallocate bits)
- 3 – Auto-variance AQ mod1, with bias towards dark scenes and grainy scenes. Recommended as grainy and dark scenes generally requires a higher bitrate to maintain quality.

- 4 – Auto-variance AQ mod2 (experimental, improves algorithm of aq-mode of 2).

Default: 1
Recommended: 3

A.4.8 --aq-strength <float>

Sets the amount of bias towards low detail ('flat') macroblocks. Reduces blocking and blurring. A higher value is recommended for grainy or animated sources (1.3-1.5).[18–20;23]

Default: 1.0
Recommended: 1.3

A.4.9 --qcomp <float>

Quantizer curve compression factor. A value of 0.0 = Constant bitrate, 1.0 = Constant Quantizer. qcomp trades off the ratio of bits allocated to 'expensive' or high-motion vs 'cheap' low-motion frames. A qcomp of 0.0 would make the quality of high-motion scenes look terrible which low-motion scenes would look great but with wasted bitrate. With a qcomp of 1.0, high-motion scenes would look great however a lot of bitrate is wasted on each frame as well since these scenes have a lot of changing frames.[18–20]

Values: 0.0-1.0
Default: 0.60
Recommended: 0.60

A.5 Analysis

A.5.1 --direct <string>

Sets the prediction mode for 'direct' motion vectors. Direct prediction tells x264 what method to use when applying motion estimation for certain parts of a B-frame.[18–20]

Values:

- none – disables direct motion vectors (not recommended, waste bits and looks worse).
- spatial – Tells x264 to predict motion based on other parts of the same frame.
- temporal – predicts motion based on the following P-frame.
- auto – switches between prediction modes depending on which performed the best so far.

Default: spatial
Recommended: auto

A.5.2 --weightp <integer>

Enables the use of weighted prediction to improve compression in P-frames. Also improves quality in fades.[19]

Values:

- 0 – Disabled
- 1 – Simple: fade analysis, but no reference duplication
- 2 – Smart: fade analysis and reference duplication

Default: 2
Recommended: 2

A.5.3 --me <string>

Sets the full-pixel motion estimation method.[18–20]

Values:

- dia – (diamond) the simplest search starting at the best predictor and checks the motion vectors at a pixel upwards, left, down, and to the right. Picks the best and repeats the process until it no longer finds any better vector.
- hex – (hexagon) similar to dia but uses a 2-range search of 6 surrounding pixels. Considerably more efficient than dia and hardly any slower, a good choice for general encoding.
- umh – (uneven multi-hex) considerably slower than hex but searches a complex multi-hexagon pattern in order to avoid missing harder to find motion vectors.
- esa – (exhaustive) an intelligent search of the entire motion space within merange of the best predictor. Similar to a brute force method. Considerably slower than umh and without much benefit.
- tesa – (transformed exhaustive) algorithm that attempts to appromiate the effect of running a Hadamard transform comparison at each motion vector, like exhaustive but better and slower.

Default: hex
Recommended: hex, umh

A.5.4 --merange <integer>

Sets the max range of the motion search in pixels for the next frame. A high --merange (>64) is unlikely to find any new motion vectors that are useful as the

search will start at the predicted motion vector so 'good' motion vectors are usually found close to the predicted one.[18–20]

Default: 16

Recommended: 16, 16+ (umh, esa)

A.5.5 --subme <integer>

Sets the subpixel estimation complexity. Higher values are better as it reduces artifacts and retains fine details. Important encoding parameter, which determines which algorithms are used for subpixel motion searching and partition decision.[18–20;26]

Levels 1-5 controls the subpixel refinement strength.[20]

Level 6 enables Rate-Distortion Optimization (RDO) for mode decision which helps improve video quality in video compression. RDO levels considerably slow down encoding speed.[20]

Level 8 enables RDO for motion vectors and intra-prediction modes.[20]

Values:

- 0 – Full Pixel only
- 1 – Quarter Pixel (QPel) SAD 1 iteration
- 2 – QPel SATD 2 iteration
- 3 – Half Pixel (HPel) on MB then QPel
- 4 – Always QPel
- 5 – Multi QPel and bi-directional motion estimation
- 6 – RD mode decision on I/P frames
- 7 – RD on all frames
- 8 – RD refinement after RD mode decision on I/P frames
- 9 – RD refinement on all frames
- 10 – QP-RD (requires –-trellis=2, --aq-mode>0)
- 11 – Full RD

Default: 7

Recommended: 7-10

A.5.6 --trellis <integer>

Increases hard-edge details by increasing efficiency in data compression at a somewhat slower speed. Usually improves overall quality of encodes.[18–20]

Values:

- 0 – Disabled
- 1 – Enabled only on final encode of a macroblock
- 2 – Enabled on all mode decisions

Recommended: 1, 2

A.5.7 --psy-rd <float>:<float>

The first value is the Psy-RD strength, representing the amount of bias in favor of detail retention. The second value is psy-trellis. The higher the value, the more detail and sharpness can be retained but also increases the risk of unwanted artifacts.[18–20]

psy-rd strength (requires --subme>=6)
psy-trellis strength (requires --trellis>=1)
Default: 1.0:0.0
Recommended: psy-rd strength = 1.0 (film), 0.4 (animation); psy-trellis = 0.6

A.5.8 --no-fast-pskip

Disables early skip detection on P-frames. Disabling pskip detection increases encoding speed but may cause artifacts in areas of solid color or gradients such as dark scenes or sky.[18–20]

Default: not set (fast-pskip enabled)
Recommended: set (fast-pskip disabled)

A.6 Ouput

A.6.1 --level <string>

Sets the level flag in the output (as defined by Annex A of the H.264 standard). 4.1 is the usual maximum value and is the Blu-ray/HD DVD standard. If not set, x264 will attempt to auto-detect the level and may not be accurate. Level 4.1 is the current Blu-ray standard and is the maximum generally supported by most devices.[15;18;19]

Note that x264 only treats this setting as an output flag, and will not take any measures to ensure compliance with the level specified.

Values: 1, 1b, 1.1, 1.2, 1.3, 2, 2.1, 2.2, 3, 3.1, 3.2, 4, 4.1, 4.2, 5, 5.1, 5.2
Default: not set (auto-detect)
Recommended: 4.1

A.6.2 --threads <integer>

Enables multi-threading to increase encoding speed. The quality loss from multiple threads is negligible unless using about 16 threads.[18;19]

Default: <1.5*logical cores>
Recommended: <logical processors>

A.7 Filtering

A.7.1 --video-filter, --vf <filter>

resize:[width=<integer>,height=<integer>][,method=<string>]
Resizes the video using the specified method to the set width and height.[27]

method: fastbilinear, bilinear, bicubic, experimental, point, area, bicublin, gauss, sinc, lanczos, spline

Default: bicubic

Recommended: bicubic or gauss, bilinear (downscaling) or cubic, spline, lanczos (upscaling)

References

[1] "x264". *VideoLAN*. 25 February 2015. Web. 9 June 2015.

[2] *HandBrake*. 8 March 2015. Web. 4 June 2015.

[3] *FFmpeg*. 16 March 2015. Web. 4 June 2015.

[4] "Advanced Audio Coding". *Wikipedia*. 24 April 2015. Web. 9 June 2015.

[5] "MeGUI". *SourceForge*. 29 March 2015. Web. 4 June 2015.

[6] astrataro. 天山牛棚. 14 March 2014. Web. 4 June 2015.

[7] "Nero AAC Codec". *Nero*. 17 December 2009Web. 4 June 2015.

[8] "Nero AAC." hydrogenaudio. 25 May 2015. Web. 4 June 2015.

[9] Bunkus, Moritz. *MKVToolNix*. 10 May 2015. Web. 4 June 2015.

[10] "Matroska Media Container". *Matroska*. 4 January 2015. Web. 9 June 2015.

[11] *Python*. 7 June 2015. Web. 9 June 2015

[12] "What is PyQt?". *Riverbank*. 6 May 2015. Web. 4 June 2015.

[13] Duquesnoy, Colin. *QDarkStyleSheet*. GitHub, Inc. 28 May 2015. Web. 4 June 2015.

[14] *MediaInfo*. Tektronix. 25 May 2015. Web. 4 June 2015.

[15] Sonnati, Fabio. "A primer to H.264 levels and profiles." *Video Encoding & Streaming Technologies*. 25 October 2008. Web. 4 June 2015.

[16] "Frame Types." *MultimediaWiki*. 4 June 2007. Web. 4 June 2015.

[17] -i-. "GOP's:I, P, and B frames explained..." *Doom9's Forum*. 6 March 2002. Web. 4 June 2015.

[18] "MeGUI/x264 Settings." *Wikibooks*. 22 May 2015. Web. 4 June 2015.

[19] "H.264 encoding guide." *Avidemux*. 11 November 2012. Web. 4 June 2015.

[20] "x264 FFmpeg Options Guide." *Linux Encoding*. Web. 4 June 2015.

[21] *.mp4 guy. "How To Use Mpeg4 AVC Deblocking Effectively (Small FAQ)." *Doom9's Forum*. 8 April 2008. Web. 4 June 2015.

[22] Werner, Robitza. "CRF Guide." *slhck*. Web. 4 June 2015.

[23] zeust. "Encoding Settings - Tips." *Hi10 Anime*. 5 July 2013. Web. 4 June 2015.

[24] "CBR/ABR/VBR: The 3 Encoding Modes." *The Complete MP3 Manager*. X-treem Software. Web. 4 June 2015.

[25] sophisticles. "Anyone try x264's new AQ mode?" *VideoHelp Forum*. 27 December 2014. Web. 4 June 2015.

[26] Botha, JJ. "X264 - Subme - Comparison." *Multimedia Mash*. 2 March 2013. Web. 4 June 2015.

[27] lutinor. "What resize method to use in x264 cli ?" *Doom9's Forum*. 13 April 2011. Web. 4 June 2015.