CSAM SUPER

A GENETIC VECTOR QUANTIZATION HILL CLIMBING VIDEO/IMAGE ENCODER FOR THE C64

BY ALGORITHM / ALGOTECH (C) 2014

WHAT IS IT?

CSAM Super is the 4th version of my Image/Video compression tool specifically designed for fast decoding on the C64.

It utilises several state of the art algorithms all which have been developed by myself (with the exception of the LBG clustering algorithm that can be initially used by the tool to 'speed up the final 'solution'

It is a VQ based encoder (Vector quantization). The encoder collates all 8x8 blocks in the whole image/animation and choses 256 blocks (or lower) that in theory would best represent the whole image or animation.

How would the blocks be chosen? Well, this is still ongoing research with hundreds of variations developed by various research labs. Before I describe how this program choses these blocks, Below is the summary of the previous CSAM versions and the methods that they had used to chose these blocks.

CSAM V1.
This application was created and used to produce the Video in my previous 'Sabrina demo' that was released sometime in 2006/2007. It does not have any 'intelligent' options for selection. The main core of the application is via user intervention where the user will have to manually chose blocks from the animation/image until the result is of a 'decent' quality. There is no clustering in use and each block is a direct 1:1 copy

CSAM V2
This was far more advanced than the previous, but some of its features were half finished. It utilises clustering options (LBG) allowing less user intervention. Quality leaves a lot to be desired

CSAM V3
This introduces the PNN (Nearest neighbour clustering algorithm) in combination with LBG clustering. Results are higher than the previous version

Now onto **CSAM V4 (CSAM SUPER)..**
PLEASE READ THE BELOW, IT WILL GIVE MORE OF AN UNDERSTANDING OF THE OPERATION OF THE PROGRAM!

CSAM Super utilises optional initial LBG clustering, but improves upon this far more by utilising mutation based Hill climbing.

Brief description of the algorithm as below..

256 (or a user defined) amount of 8x8 blocks are chosen at random from the entire image/animation. To speed up later encoding and to increase the quality of the random selection, only original 8x8 blocks (different) are used. Why 8x8? This is due to the c64's built in tile capabilities that can plot an 8x8 block by one byte write.

To ensure that each block can resemble more of the image, LBG clustering is then used. LBG clustering reads each 8x8 block in the origjnal image/animation and clusters around the codevector (those 256 blocks that were chosen randomly) that is most closest to it.

Once all blocks in the source image/animation have been read, the clusters are merged to form a new set of 256 blocks. You will now find that most of these individual blocks can best resemble more of the sections in the image/animation.

There is a weakness in this method however. It will be very common that there will be a considerable amount of codevectors that the source blocks did not cluster with at all. This increases the error of the image as it will in most cases only resemble a 8x8 block in the image.
This is beneficial for important area's of the image, but too many non clustered blocks will cause issues. (See Figure 1.1)

FIGURE 1.1. IMAGE GENERATED BY LBG

This is where ELBG (Enhanced LBG) can be used which identifies these blocks and shifts these in order to allow merging. Quality increase is rather significant (Figure 1.2)

FIGURE 1.2. IMAGE GENERATED BY ELBG

CSAM Super uses a completely different and more effective approach to identify blocks and to chose more optimum ones.

Once CSAM has LBG merged the blocks, it then starts a heuristic based 'blind approach' that picks 8x8 blocks from the source image at random and maps it to a random section in the codebook.
For each iteration, the error is analysed and worse errors result in 'undo-ing' of the modification.

What this tends to do after many iterations is that it removes the 'weak' 8x8 blocks in the codebook and replaces it with more optimum ones.

LBG can then be run again and this normally results in less errors due to more merging.

Just this process along results in quality far more superior than LBG with substantial improvements over ELBG/PNN

The encoder can then also apply various other options for codebook selection such as

Mutation mode closest
A new member is introduced into the population and he immediately merges himself with a female that is most closest to his 'personality' Before this is finalised, they both check whether or not, they have made an improvement. If so, they live for the time being, otherwise they split from each other. She stays and the male leaves

Mutation mode blind
A new member is introduced into the population, the difference here however is that he is greedy and decides to merge with the first female he see's

Pixel Mutation mode
A small splash of color is randomly placed at a random pixel in a codevector. This mode should only be used in the final error reduction of the codebook.

Furthermore unlike ELBG and other codebook generation methods, there is the option to use the standard 8x8 mode as well as 8x4 and 4x4 mode (This uses a 4x4, 8x4 section from anywhere in the source animation image and merges itself to an edge of a particular codebook. There is also the option of sliding 8x8 selection which does not restrict the grabbing of a codevector from fixed 8x8 offsets in the source (this reduces errors significantly (See Figure 1.3)


FIGURE 1.3 (IMAGE GENERATED BY CSAM SUPER)

The encoder can also weigh specified blocks more or less than others. This data can be manually drawn over the preview image for ease of use as well as saving and loading the mask data.

This ensures that important sections (for the user) are prioritized more than non important sections. (For example, there could be an image of a someones face with lots of hair. The VQ encoder would allocate just as much importance to the hair, leaving the other face sections of equal quality). Applying the weighting sections can result in an image like the below (FIGURE 1.4)


FIGURE 1.4 (IMAGE GENERATED BY CSAM SUPER – WEIGHT MODE)

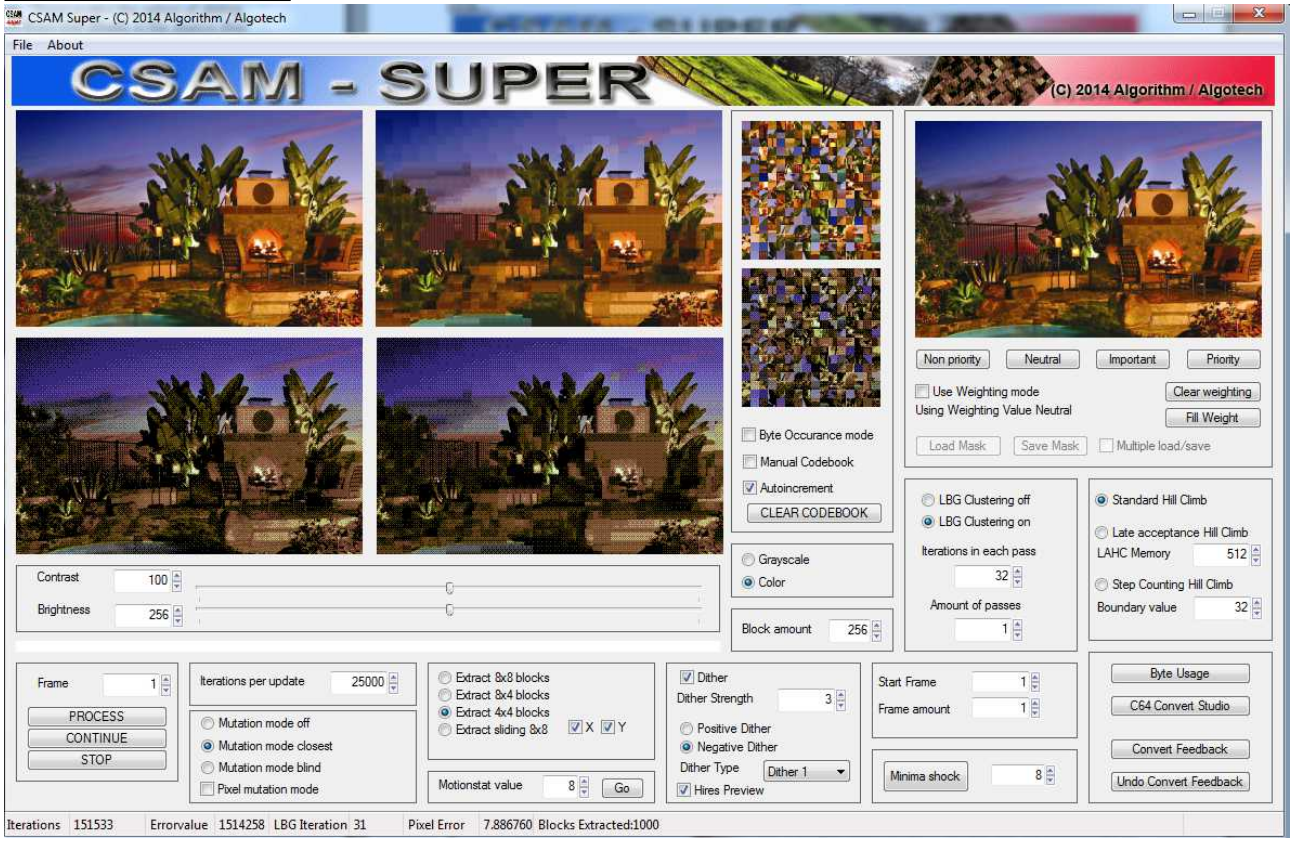
There were a few initial obstacles which caused an issue and they were mainly related to the initial version of the VQ decode. As mentioned previously, A VQ decode takes place after each codebook merge/selection to determine whether to keep the selection or discard it. This has to be done many tens of thousands of times. Luckily i am using a lazy VQ encode method which is just as optimal and this results in a major speedup.

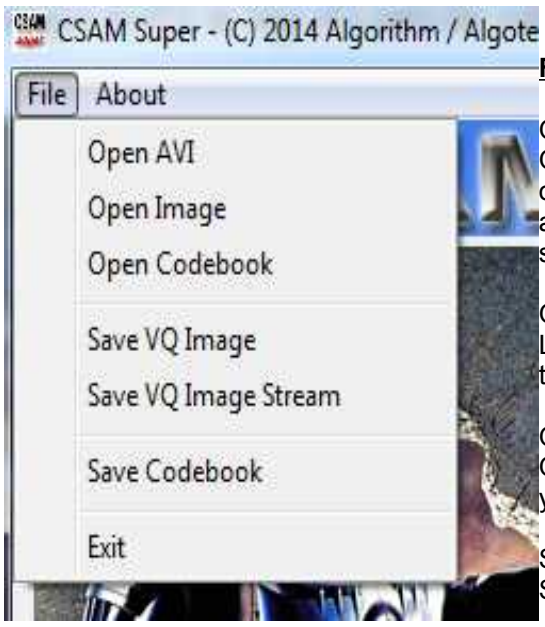
There is also the limitation of the 'hill climbing' approach hitting a local minimum. (This is where there needs to be a substantial reorganising and change in codevectors in order to improve the results.

This is where i have used two additional hill climbing methods which take far longer but normally create more optimum final results (Late Acceptance hill climbing and Step count hill climbing (More on that later). There is also the option of manually forcing worse results in order to try and escape from the local minima issue.

Now onto the application itself.

THE APPLICATION





FILE LOAD / SAVE

OPEN AVI
Open and load an AVI file. If the video is larger than 320x200 or has more than 256 frames, it will use the first 256 frames and crop to 320x200 from the top left hand corner. You MUST set the start frame and number of frames before.

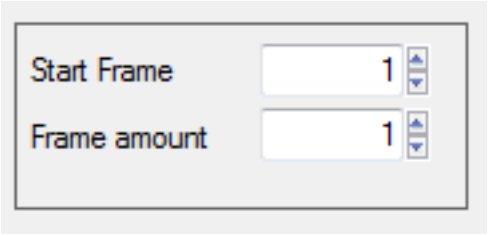
OPEN IMAGE
Loads a bitmap image. If the image is larger than 320x200, then it crops the image to 320x200.

OPEN CODEBOOK
Open a previously saved codebook (To continue from where you left off)

SAVE VQ IMAGE
Save the current converted truecolor image

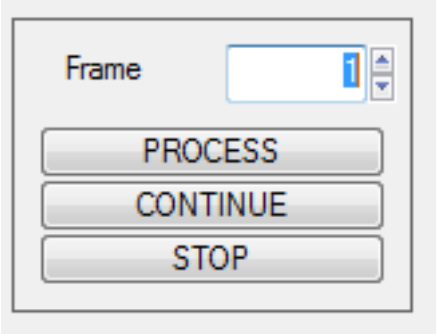
SAVE CODEBOOK
Save the generated codebook

START FRAME / FRAME AMOUNT



This is where you SPECIFY where the image/animations will be loaded to (as well as the amount of frames.) This needs to be set beforehand, otherwise the animation will only be loaded as one frame (or the amount of frames specified) Another thing to bear in mind is that the Encoder will ONLY operate on these frames. If there is any data that is outside this range, the VQ encoder will still make an attempt to encode the data, but quality for these outside frames will suffer.

PREVIEW FRAME / PROCESS / CONTINUE / STOP

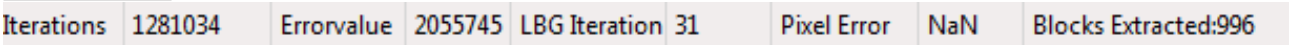


Adjusting the frames let you preview the frames that are encoded.
PROCESS – This option starts encoding from scratch (and generates a new random codebook) As the mutation / hill climbing operates in a random nature, you will find that quality increase may occur on several runs.

CONTINUE - This allows the encoder to continue without changing the codebook that may have previously been generated. Options can be tweaked

STOP – The encoder runs forever. To stop the process, press the stop button

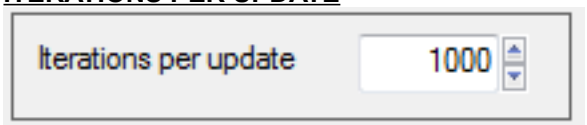
STATUS BAR



This is very useful and allows the user to see whether or not there is any improvement that is made to the image. (Errorvalue. The lower the number, the 'closer' it is to the source image as a whole)

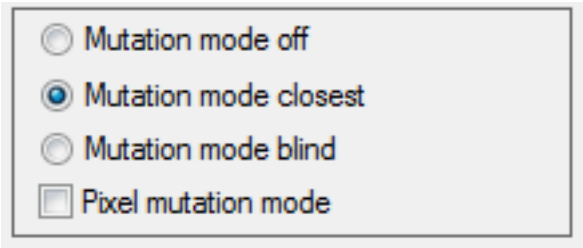
As mentioned previously, the settings in the encoder can be changed while the data is being encoded, Pay attention to the 'errorvalue'

ITERATIONS PER UPDATE



The lower the number, the quicker the user see's any changes on the screen. (Recommended to keep this number higher than 1000) - This is also linked to the counter that activates the LBG (Will change this in a future release) It is recommended to initially use an iteration value of around 25000 (default). This will result in the image being encoded with more positive changes before any future lbg encode. A lower value will mean that the lbg encode will be activated after only x iterations. This does not apply if LBG mode is turned off.

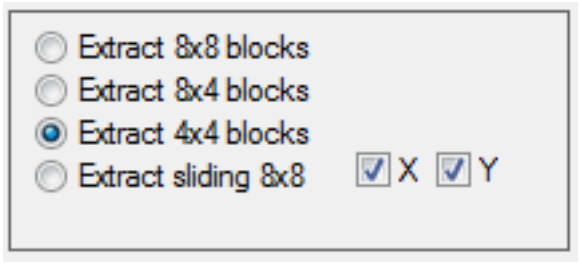
MUTATION MODES



Determines what to do with the 8x8 block that is chosen from the source image. It is always recommended to keep this option on 'off' initially. When there is less improvement in the error, then to adjust while running to the other modes.

Mutation mode off – Direct copy and comparison from source image / animation to codebook
Mutation mode closest – Merge with closest codebook entry
Mutation mode blind – Faster than the above, but least likely to produce a positive outcome
Pixel mutation mode – turns off extract mode and randomly places a random pixel color in a random area of a random codevector. Can reduce errors further but only recommended as the final encode step (works better for grayscale images)

EXTRACT MODES

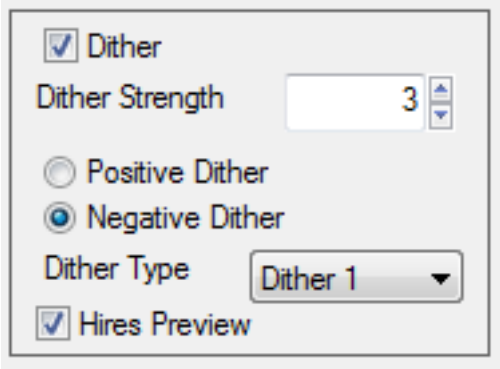


These options are used together with the mutation modes. Extracting 8x4 or 4x4 blocks into a specific corner of a codebook takes longer, but results in higher quality. Recommended initially to use 8x8 (with mutation mode off) Then when there are less error changes, to select mutation closest. Then 8x4 and 4x4 until error changes are less.
The extract sliding 8x8 is useful for animation sequences where there may be panning of the image up/down/left/right this can average out panned blocks. The user can also specify whether the sliding mode should be in x or/and y. If both these are unchecked, it has the same purpose as 'extract 8x8blocks'

MOTION STATIC

This option is useful only in video mode. It ensures that slightly moving sections (eg background) remain static which helps in quality and compression of individual frames – eg delta. Higher values result in more sections remaining static. If you are not happy with the result, select '0' and run motion static to reset. (or use a lower number) You should only use this option when the encoder is 'stopped' You may preview the results by changing the preview frames to go back/forwards between each frame.

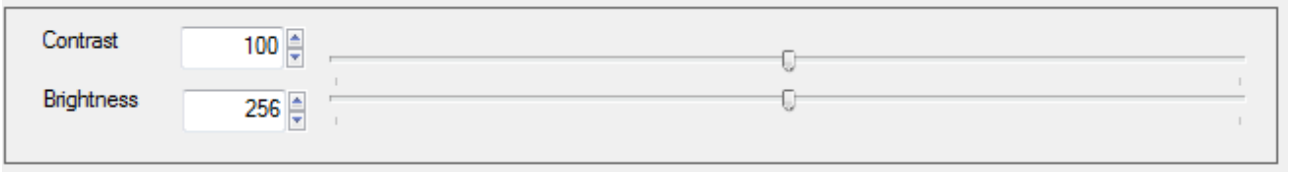
DITHER



Can be used for preview purposes (to give a rough insight of how the image may look like on the c64) The main GUI has 4 images (Top Left is the original) – (Top right is the original VQ encoded) – (Bottom left is the C64 color conversion from the original image) – (Bottom right is the C64 color conversion of the VQ encoded

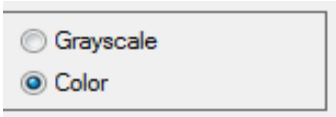
image) – To see the actual quality of how it would look like on a C64, the 'C64 studio' Option must be used. (More on that later) – The settings above do not need any explaining. This option together with the brightness and contrast gives a quick preview of how the final image may look like (Its a good idea to adjust these together with the brightness and contrast before encoding)
Another major feature is the ability to feedback the converted image into the source (more on this later) For this reason, I have included a lores and hires preview option.

CONTRAST / BRIGHTNESS



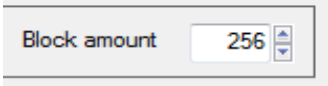
This option should only be used before encoding (It can be used while encoding, but this would result in the VQ encoder having to reencode the data (as the source data will have changed) – Furthermore, it may result in the VQ encoder not able to improve any further – As the error rate after the change cannot result in an improvement by just one codevector update (If this is the case, then stop the encoder and continue – Which recalculates the errors)

COLOR MODE



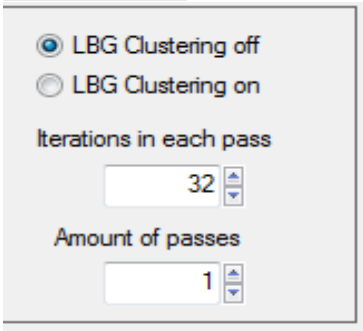
Specify whether to encode in color or grayscale (Color encoding takes more time and image quality can suffer even more than grayscale) due to the requirement of more image blocks in color to have similar quality as it would do in grayscale. When changing the mode, the codebook is deleted and the data has to be recalculated.

BLOCKAMOUNT



Default is at 256 (recommended) Using a lower amount reduces the quality but may be required in the case of perhaps requirement (eg if encoding for Atari that can only use 128 chars in char mode)

LBG OPTIONS

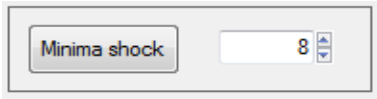


Specifies whether or not LBG clustering will be used alongside the Genetic Hill climbing method. It is recommended to leave this on. However if stopping the encoder, this method can later be switched off and then the continue option can be used to tweak codebooks without having to run the lbg again.

The amount of passes indicate how many times the LBG will run. This is used alongside the 'Iterations per update option' eg. If the amount of passes is '3'. And iterations per update is 10,000. The LBG will run initially when the encoder is started, and then after 10,000 changes, it will run it again for x amount of times.

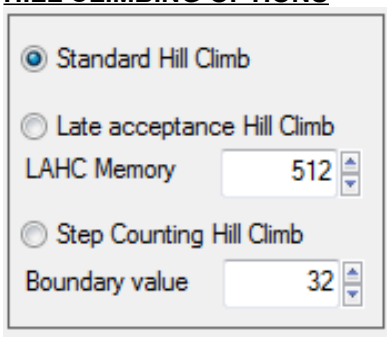
The iterations in each pass option is the amount of times that the LBG will iterate. Normally not much point setting this to a value above 30.

MINIMA SHOCK



This option is useful while the data is being encoded to 'force' more errors in the codevectors. This can give better end results due to escaping some local minima conditions. The user can specify the strength of the 'shock'

HILL CLIMBING OPTIONS



The encoder utilises Hill climbing in combination with the LBG and mutation approach. Information on the hill climbing modes as below

Standard Hill climb – If encoded results are better than previous then keep, otherwise discard
This is the fastest option and normally should be the one to use if time is an issue (In combination with the mutation and block extract options, it will result in a high quality codebook)

Late acceptance Hill climb. If encoded results are better then previous, keep, if not, then compare error to previous x (lahc) steps back, if lower then keep, otherwise discard
Recommended minimum LAHC memory number is 512. This method allows worsening moves which prevent the encoding hitting a local minimum

Step counting hill Climb – If encoded results are worse then the saved error, then discard, otherwise accept and increase counter. Do not update error and keep the saved error, until the counter has been reached. Then update the error value
This in most cases is even more effective than the Late acceptance hill climbing method. Again this option as well as the LAHC take far far longer to reach acceptable quality, but normally results in less errors.

CODEBOOK OPTIONS

Byte Occurance mode – When this is activated, it allows the user to select a codevector (original or c64 preview) and it displays where the codevectors are located on the encoded image – Can be useful.

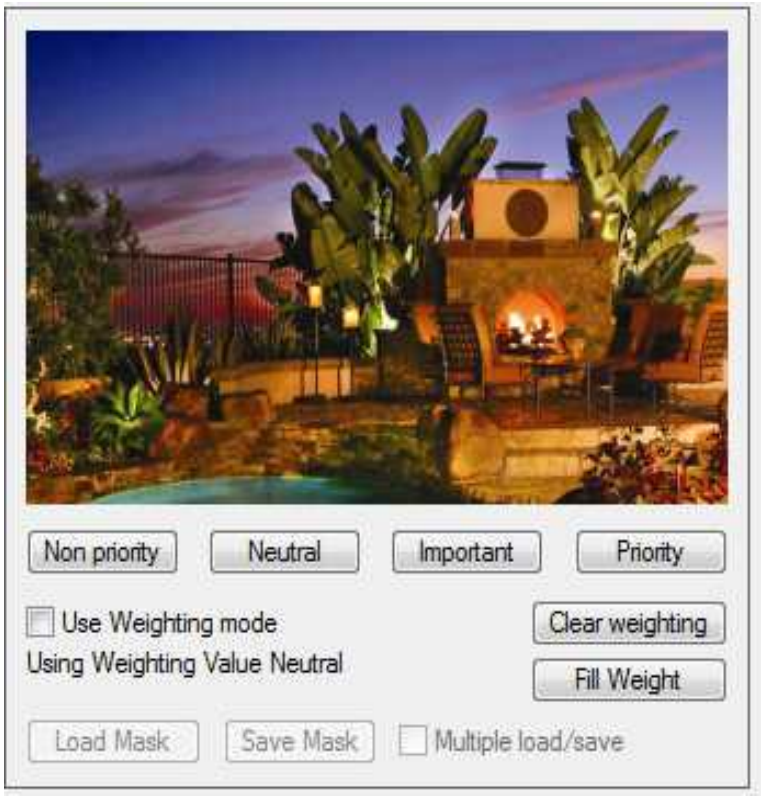
Manual Codebook – When this is activated, the user can use the mouse over any of the preview images and click on a section to save the 8x8 block into the codebook table. A codebook to replace can be selected as well.

Autoincrement – Instead of constantly going backwards and forwards between the source image and the codebook, this increments the codebook counter allowing the user to 'paint' away.

These above options can be used if wanting to override codebook values. (As with all the other options, updates are in realtime)

Please note that there is a possibility of the encoder not able to reduce the errors if many successive changes have been made to the codebook (if this is the case, then stop the encoder, Turn off/on lbg if required, and then continue the encoding which recalculates the errors.

WEIGHTING OPTIONS



A powerful tool which can dramatically increase the quality of the result. The encoder attempts to reduce the whole total error of the source image/animation in comparison to the result that is constantly being encoded. Unfortunately it has no way of knowing which sections are more important than others (an exception can be made if utlising face tracking, which may make it in a future version)

The weighting option when turned on allows the user to paint over sections that are non important, neutral (default), important or very important. This can be done frame by frame (by using the current frame toggle to switch between the frames)

Select the required priority button and then paint over the sections. This should be done when the encoder is stopped as it will more than likely result in the encoder not being able to make any improvement until recalculation (via stop button and continue)

The error amount will be higher, but the percieved error (if the weighting options are used properly) will be less.

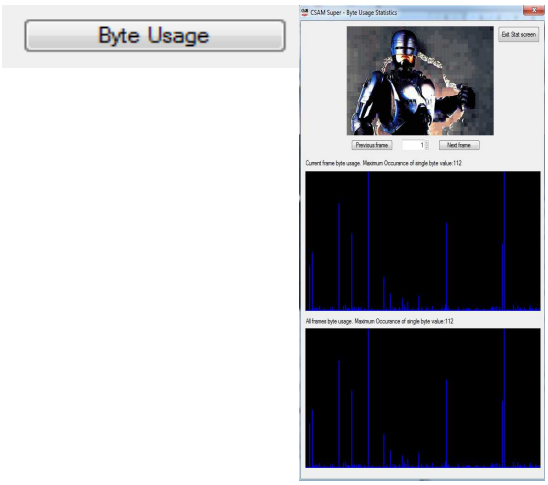
If you find that the important sections are still not being encoded properly, then use the non priority option to paint over sections that are not important. This will start the reassigning procedure.

Again as noted previously, for initial fast results, always ensure that mutation mode is turned off initially and 8x8 mode is selected. Then gradually apply the other mutation modes while encoding.

Weighting data can also be loaded and saved. To save the current frame only, make sure the multiple load/save data is unticked. Loading weight data when this option is unticked loads the weight data into the current displayed frame.

If the multiple load and save option is enabled, then the weight data is loaded/saved using the values in start frame and frameamount

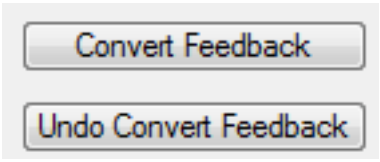
BYTE USAGE



Selecting this option gives some statistical information on the whole animation and individual image that can be previewed and updated in real time.

Please be patient if there are many frames that are selected. It may take a while for the screen to be displayed.

SELF FEEDBACK



Another powerful option here. This enables the c64 previews (from start frame to frameamount) to be fed back into the encoder as source data.

This can increase the quality of the final result (but also at the same time, make the encoder less efficient due to low resolution color data)

One example would be to adjust brightness and contrast until the image is 'simplified' eg dark area;s turned to black etc. When the convert feedback button is used, the encoder then has less blocks to deal with. Please ensure that the dither options are then turned off (from both the c64 studio options as well as the preview.

If the feedback to source was in 'lores' mode, then ensure that only the lores options are used in c64 studio and vice versa otherwise quality will suffer.

C64 STUDIO

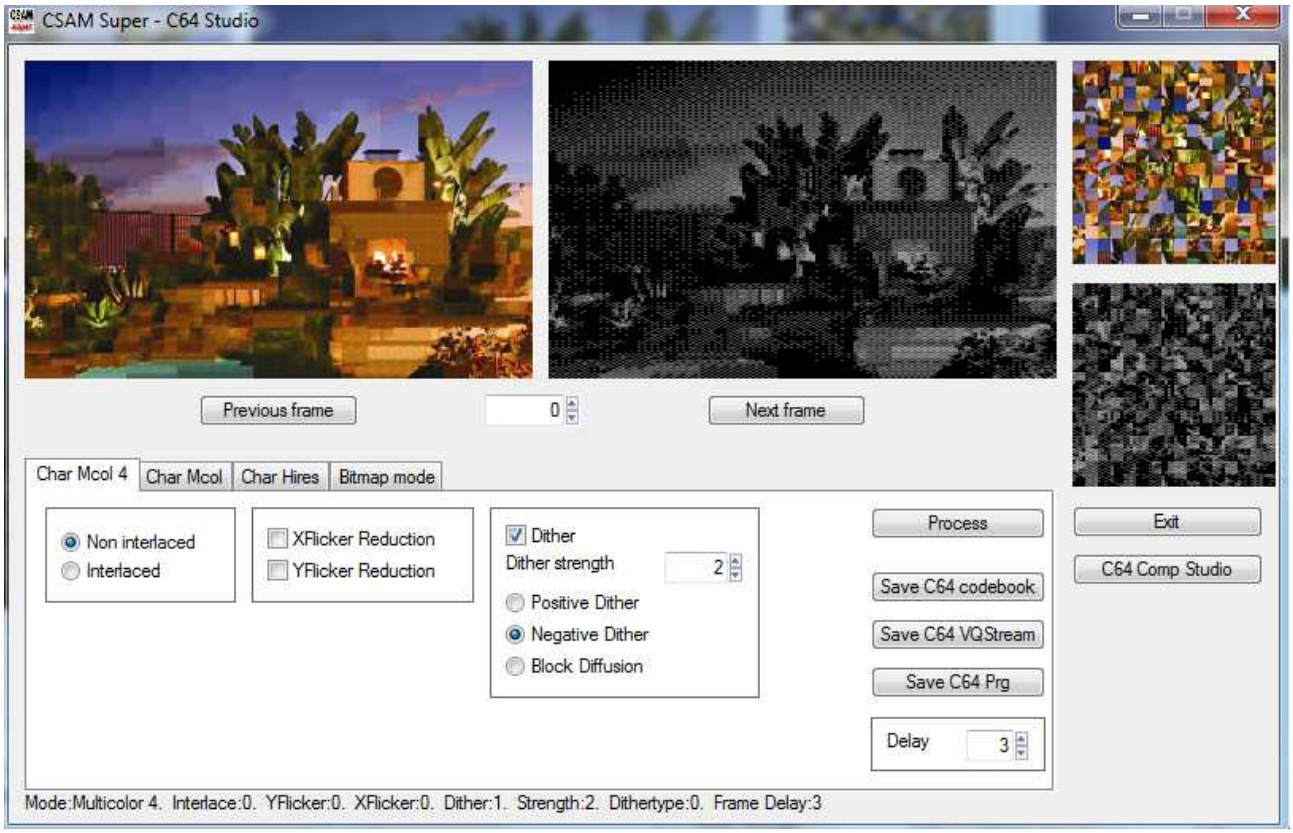
This section is the section to use once you are happy with the encode results. This module allows the user to save the data to c64 format including raw VQ frames (1k each frame), codebook data (in c64 format) as well as even self runnable C64 files. There is also the ability to compress frames more efficiently as well as optimizing lookup table data and codebooks from within this gui.

Self runnable C64 files in this module have a limit of around 40 frames (and around 34 frames in bitmap mode) This will be increased when the 'Compression studio' has been implemented in CSAM Super.

Changes are ONLY made when the process button is pressed

Some information on the modes as below.

MCOL4



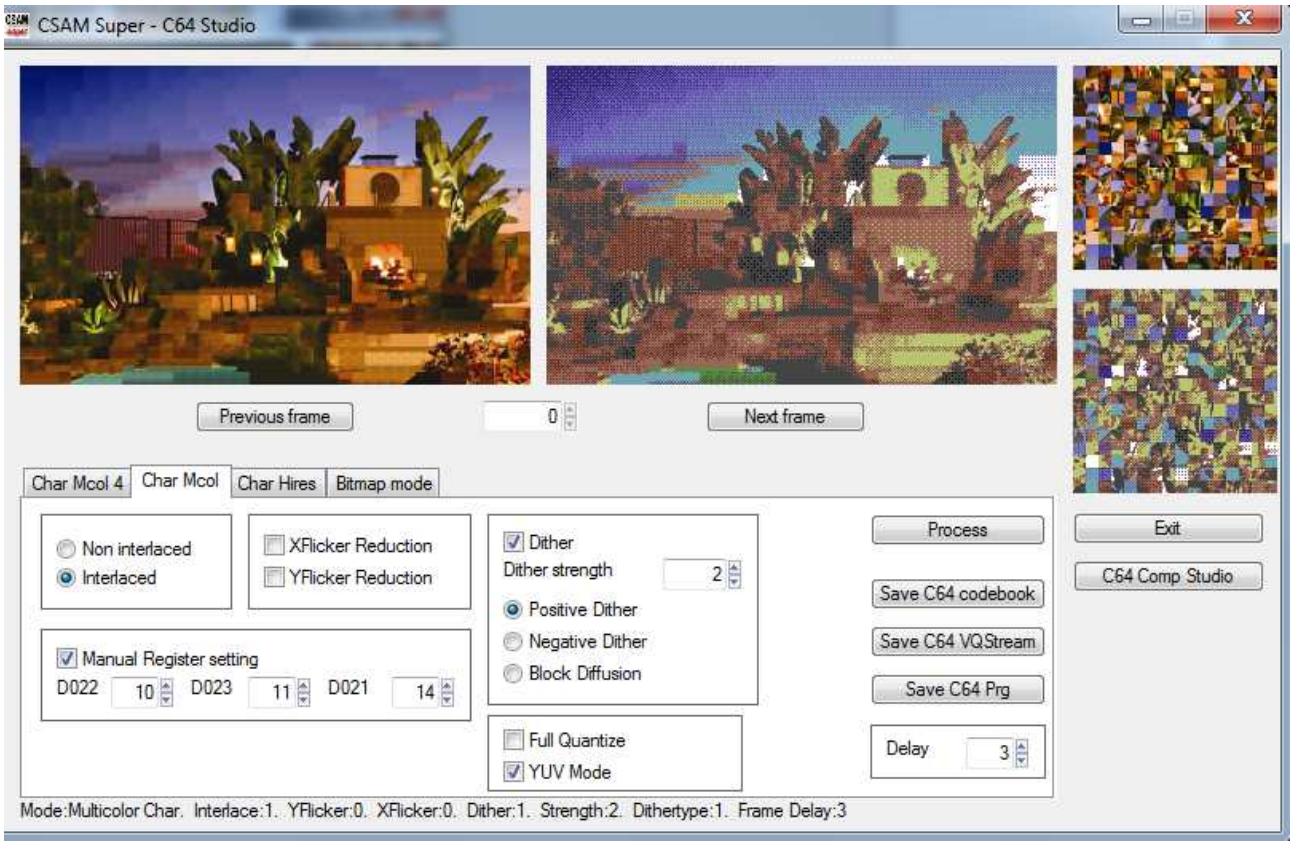
Converts the video/image into 4 colors (standard char mode) with no additional color requirement. This is the least CPU intensive C64 mode and does not require plotting of color data.

Non Interlaced – 2048 bytes codebook data 4 colors
Interlaced – 4096 bytes codebook data. 4 colors xshifted for 320x200 'resolution'

Xflicker and Y flicker can be applied to reduce the flicker. Please bear in mind that using these options can increase the flicker. X flicker on its own tends to work better (and X and Y flicker works well for block diffusion mode (Block diffusion mode is experimental) and most of the time can give images of poor quality.

The block diffusion option works by diffusing only inside the 8x8 block which is useful for compression in this case.

MCOL CHAR



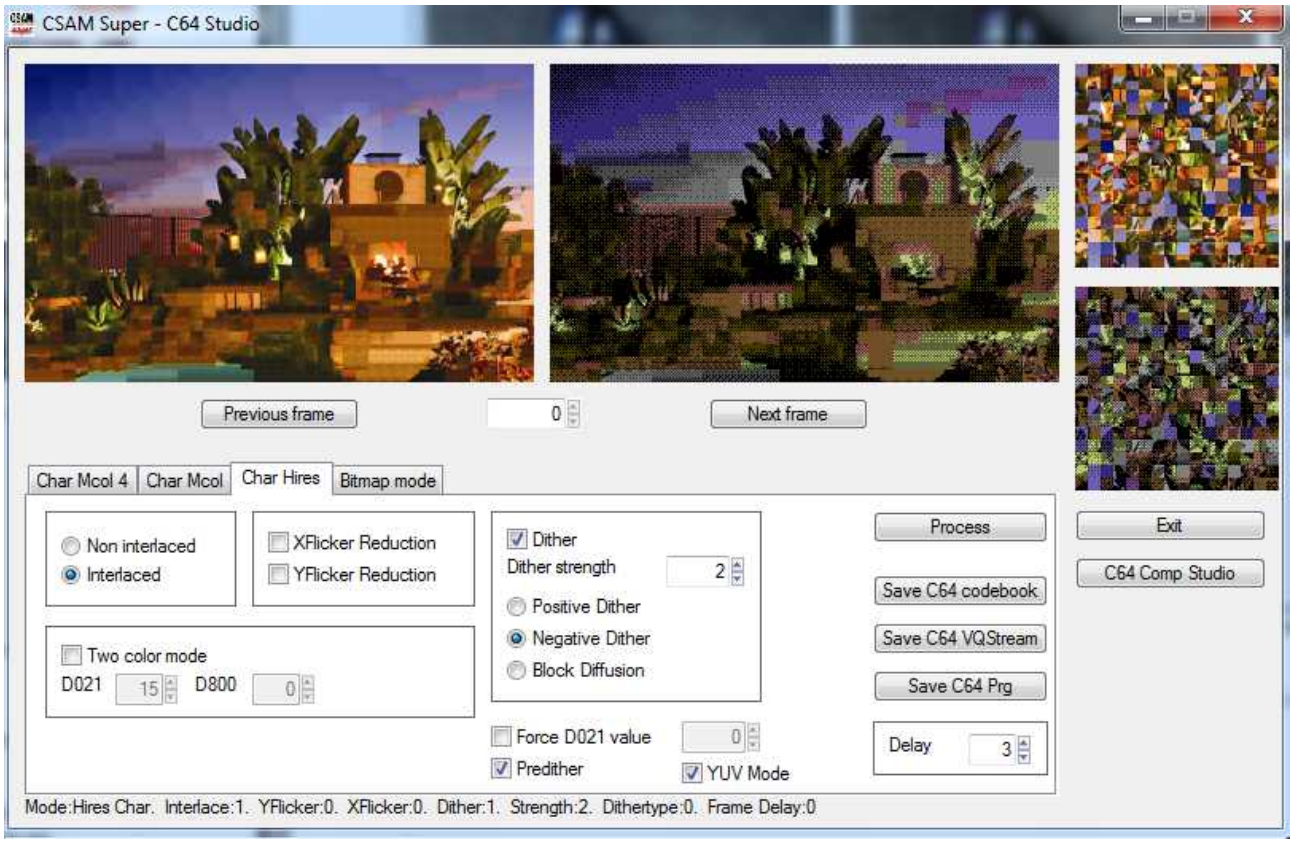
Uses standard Char mode, but this time a more complex method of color reduction is used. This variation can only have 3 different colors for the whole image (and then for each block, only an additional color (the first 8)

Non interlaced – 2048 bytes codebook
Interlaced – 4096 bytes codebook data.
D800data. 259 bytes of data with each byte having a 0-7 color and associating itself with the codebook. Bytes 256-258 are the d022, d023 and d021 data (if saving as raw)

As mentioned previously, experiment with the xflicker and yflicker options. If you are not happy with the results that the converter has created, you may override the brute force and select the color values yourself (3 universal colors)

Full quantize option dithers the colors before quantizing, while not using this option quantizes directly from the truecolor image (normally full quantize gives better results). Recommended to also turn on YUV mode.

CHAR HIRES



Hires mode that has the following options

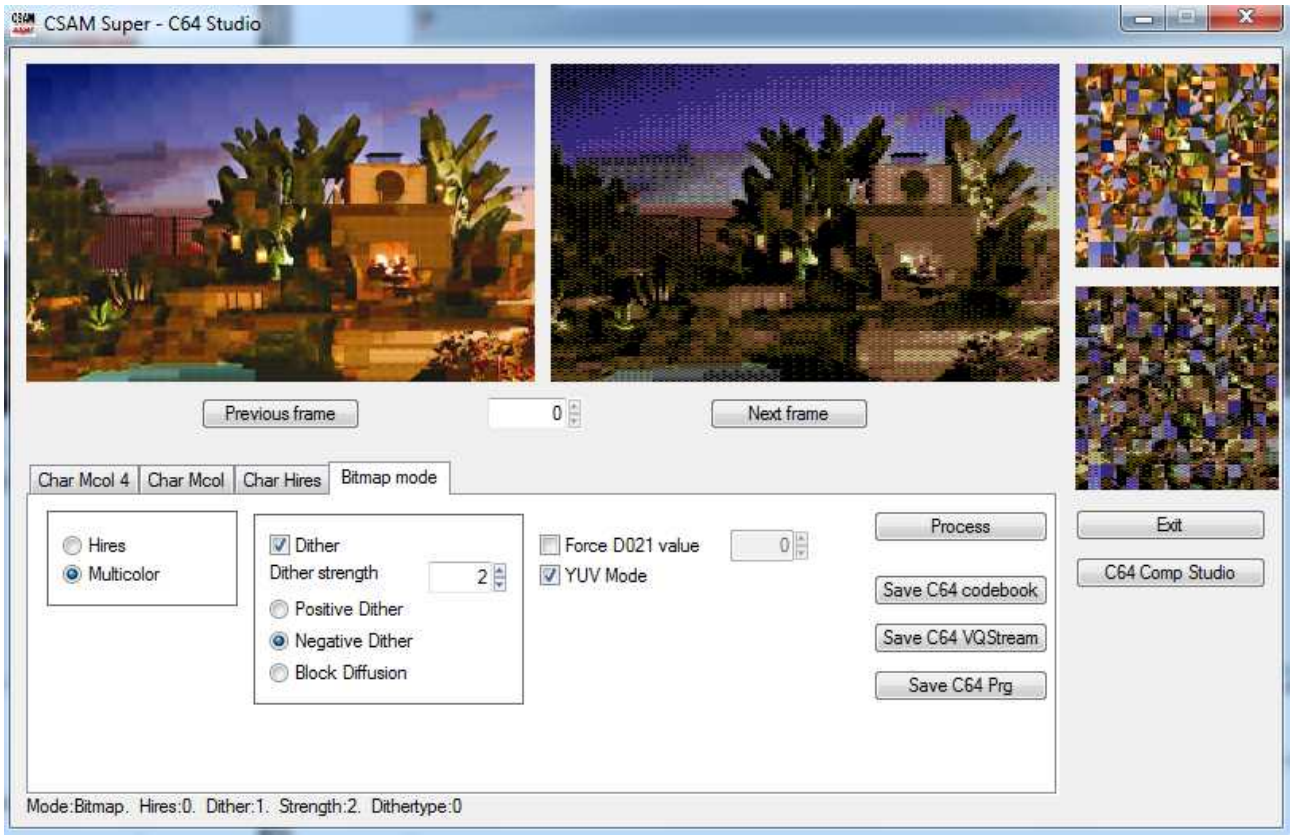
Non interlaced – Standard hires char mode
Interlaced – An additional color is produced by the mix of two colors – please note that there are no luma checks, it is down to the user to select colors that are not conflicting too much

Two color mode – When this option is on, there is no requirement for the c64 decode to plot color data. When two color mode is turned off. The brute force option is used to generate 2 colors (including an inbetween color) for each 8x8 block. This has more impact on the decoder (c64) as it has to also plot the color data for each frame – again luma is not checked. Will incorporate in a future version

Force d021 value – If results dont look too good in the non-two color mode, you can manually change this value and experiment with various values.

Pre-dither. Dither the image before quantizing. Normally gives higher quality. Also recommended to keep the YUV mode on.

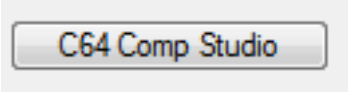
BITMAP MODE



Both Hires modes and Lores modes are brute force trying all possible combinations of d021, ink, paper, d800 for each block. To speed things up, D021 can be forced. Keep YUV mode turned on.

The C64 decoder is not optimised hence only runs at around 5 frames per second. But ok for a preview.

C64 COMPRESSION STUDIO



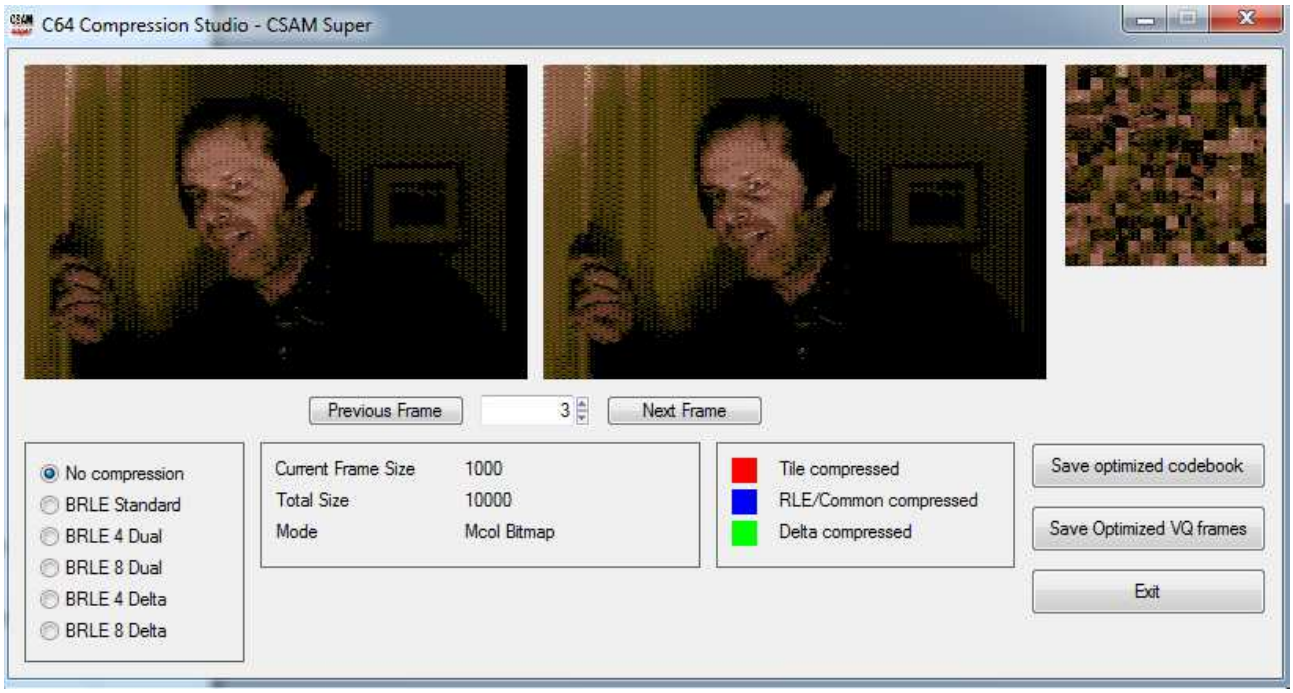
This option is available from the C64 studio and optimizes the converted codebook as well as remapping LUT data in order to make the data more compressible. Any duplicate codevectors that have been created due to quantization are removed and data is remapped accordingly.

Furthermore various frame compression options are available. These types of compression are 'light' compression that can be decoded fast on a C64 (and the data can also be compressed further with other compression methods such as LZ if required)

For information on the fileformat for these compressed data types, please read the accompanying 'compressedfileformats.txt' document.

Codebooks and optimized/compressed LUT data can be saved within this module. Information about each compressed frame size and total frame size is also available when going backwards and forwards in the animation.

NO COMPRESSION

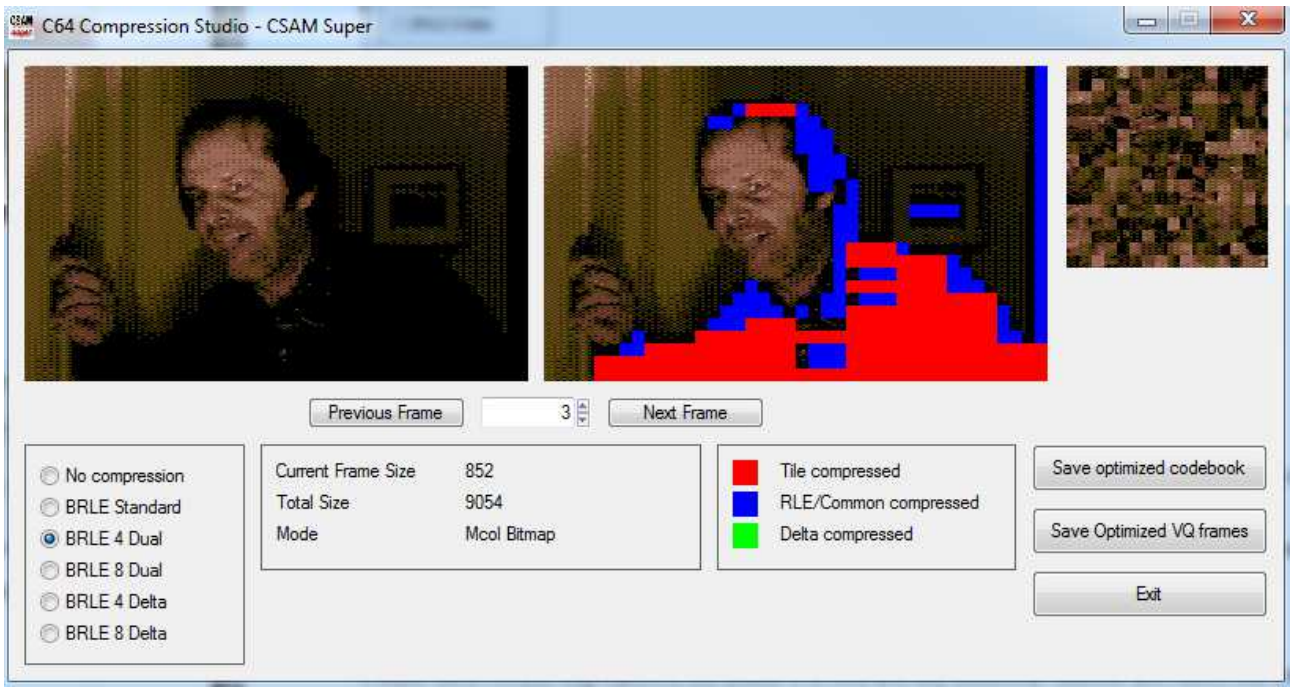


The codebook and LUT data is optimized but not compressed. This would be useful if the user wants to compress the frames using their own method.

BRLE STANDARD

Each repeating character is packed to 1bit (This is not as good as it sounds as the bitbuffer requires 128 bytes of space) Useful if there are many complex frames and more changes between each frame in comparison to the other modes.

BRLE 4/8 DUAL



Dual bit compression. If there are 4 or 8 characters of the most reoccuring byte in a frame, this is packed to 1bit. If less than this, then the 4/8 bytes in the frame are packed into 4/8 bits as well as any bytes that are different to the most reoccuring byte. This mode is mainly better suited for areas with lots of blank gaps (eg silhoette based animations.

BRLE 4/8 DELTA

Very useful for frames where there are less changes. In the above example, the moving hand in each frame is compressed to around 60 bytes per frame. (total of 20 frames compressed to 2.2k) and bear in mind that all this data including the other modes can be compressed further using other methods. Eg pack using LZ and depack the semi decompressed data, then play back.

SYSTEM REQUIREMENTS

Requires around 700mb ram and a minimum HD display (recommended 1920x1080) – May change the GUI to fit more on other PC's

There are two C64 self runnable demo's to give a rough idea of the quality thats possible with this encoder.

OTHER INFO

Please send me any bug reports (I am aware of a few) and any additional features and idea's that you may want me to possibly incorporate into this tool.

Email
thealgorithm@msn.com

Facebook
https://www.facebook.com/thealgorithm

IRC
I can sometimes be reached via IRC at #c-64 (ircnet)

CSDB.
Http://csdb.dk