Disclaimer: I am a chemist. I don't know much about video formats or media encoders. My `FFmpeg` knowledge consists of the bare minimum that I needed to make videos that didn't look like hot garbage. Take most things in this document with a very large grain of salt, and if the `FFmpeg` manual or some other part of the internet says I'm wrong, don't be surprised if I'm wrong. The Igor tool also only does the bare minimum needed to make good videos. There is much room for improvement, but that's not really what I am paid to do.

Contents

1 Background

Igor has functions to make videos (**NewMovie**, **AddMovieFrame**, etc), but the videos usually are pixelated and the file size is huge. The best way that I have found to make videos is to use Igor to make the individual frames and then use a program called `FFmpeg` to convert the frames into a movie. The movies made by this method will look way better than the movies made with Igor commands, and the file size will not be much larger than the size of an individual frame.

`FFmpeg` is a pretty widely used program, and you can find tons of information about how to use it online. If you run into issues search online for your question.

`FFmpeg` is command line only. There is a free program called OpenShot that has a GUI interface, but I have never used it. It seems to use `FFmpeg` for many things.

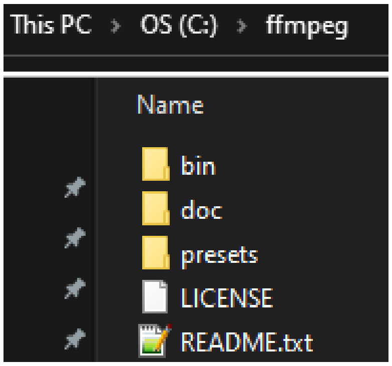
2 Installing `FFmpeg`

`FFmpeg` can be downloaded from here: `https://ffmpeg.org/download.html`. I generally use the Release build. I've also gotten the Essentials instead of the full build. I usually download the "Release build" (the stable version with a "normal" looking version number).

The Igor script will need to know the path to `FFmpeg`. There are two paths in the ipf (in Igor path format), one for Windows and the other for macOS. If you install the program in the same place that I did, then you won't have to change the ipf. If you install the program somewhere else, you will need to change the string constants at the top of the ipf to the correct paths for your setup. Make sure to change the correct string constant for the OS you are using, and no matter the operating system use Mac style separators (colons).

2.1 Windows

On Windows, I manually installed the `FFmpeg` folder in C. I downloaded the zipped file containing the release build, unzipped it, renamed the folder "ffmpeg," and moved it to C. The path to the `FFmpeg` executable is then `C:\ffmpeg\bin\ffmpeg`.



`FFmpeg` location on Win 10

2.2 macOS

For Macs, the preferred way apparently is to use HomeBrew. I don't use the admin account on my Mac for everyday use, and I ran into some issues installing HomeBrew as a non-admin user. I wasn't in the mood to find the solution to this, so I just downloaded a static build and installed it in usr/local/bin. To get to that folder, I opened Finder, pressed Command+Shift+G, and entered /usr/local/bin into the dialog box. I needed admin permission to install there, but that wasn't an issue.

2.3 Shortcuts

Once you have installed `FFmpeg`, if you want you can add it to your environment variables (Windows) or bash profile (Mac). If you do this, then you can just type `FFmpeg` into the command prompt, and the operating system will know what to do with that command. If `FFmpeg` has not been added to the environment variables/bash profile, you can still call it from the command line, but you will need to give the full path. You can find instructions for doing this in various places online (Windows, macOS). The ipf uses the full path to `FFmpeg`, so you do not need to do this for the ipf to work properly.

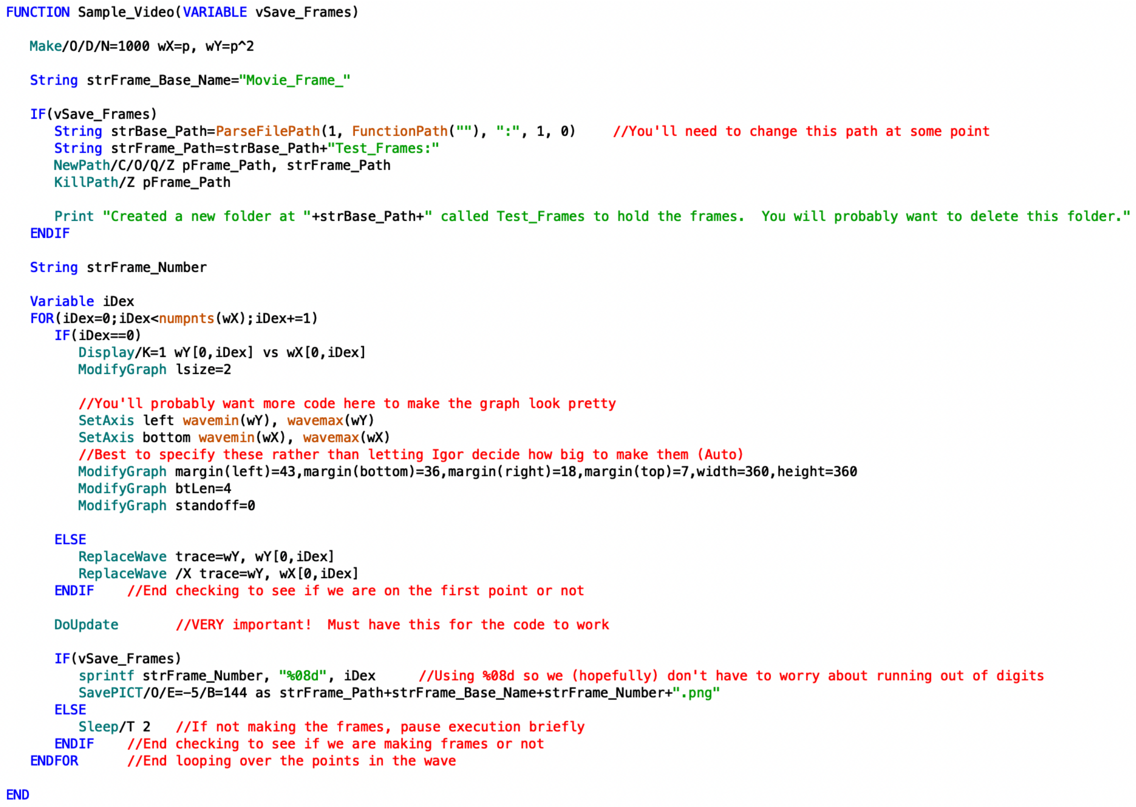
`FFmpeg` (to my knowledge) is command line only, and if you try to run the executable by double clicking on it you might not see anything. If you want to verify that it was installed correctly, type "ffmpeg" into the command prompt (or enter the full path to the executable if you haven't added it to your environment variables). If everything was installed correctly some information about the version you have installed and the various plugins should get printed to the terminal.

2.4 Updating

To update `FFmpeg` on Windows, simply delete, rename, or move the old folder, download the new version, and install it as before. I think I did something similar on macOS.

3 Making video frames

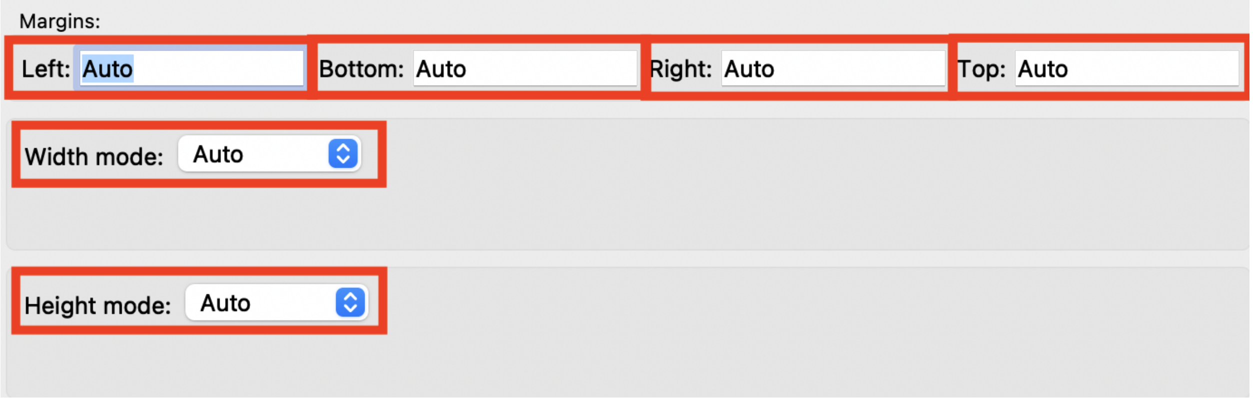
To make the videos, you will want to create a series of images that `FFmpeg` will then stitch together to make the movie. The names of the images should have an index at the end that increases by 1 for each new image. There is a command in `FFmpeg` that you can pass that doesn't require the indices to increase by 1 (`-pattern_type glob`), but it apparently doesn't work on Windows, and I've never tested it. The image names should be of the form `[Base name]_[index]`, where the index is padded with zeros so the number of characters is always the same. Sample code is shown below. Example image names would be `Image_000001`, `Image_000002`, `Image_000003`. I have not tried non-padded indices (e.g. 1, 2 , 3, instead of 000001, 000002, 000003). I also use a large number of index digits (e.g. 6 or 8) so there is no danger of running out of space.



```
FUNCTION Sample_Video(VARIABLE vSave_Frames)

   Make/O/D/N=1000 wX=p, wY=p^2

   String strFrame_Base_Name="Movie_Frame_"

   IF(vSave_Frames)
      String strBase_Path=ParseFilePath(1, FunctionPath(""), ":", 1, 0)     //You'll need to change this path at some point
      String strFrame_Path=strBase_Path+"Test_Frames:"
      NewPath/C/O/Q/Z pFrame_Path, strFrame_Path
      KillPath/Z pFrame_Path

      Print "Created a new folder at "+strBase_Path+" called Test_Frames to hold the frames.  You will probably want to delete this folder."
   ENDIF

   String strFrame_Number

   Variable iDex
   FOR(iDex=0;iDex<numpnts(wX);iDex+=1)
      IF(iDex==0)
         Display/K=1 wY[0,iDex] vs wX[0,iDex]
         ModifyGraph lsize=2

         //You'll probably want more code here to make the graph look pretty
         SetAxis left wavemin(wY), wavemax(wY)
         SetAxis bottom wavemin(wX), wavemax(wX)
         //Best to specify these rather than letting Igor decide how big to make them (Auto)
         ModifyGraph margin(left)=43,margin(bottom)=36,margin(right)=18,margin(top)=7,width=360,height=360
         ModifyGraph btLen=4
         ModifyGraph standoff=0

      ELSE
         ReplaceWave trace=wY, wY[0,iDex]
         ReplaceWave /X trace=wY, wX[0,iDex]
      ENDIF    //End checking to see if we are on the first point or not

      DoUpdate       //VERY important!  Must have this for the code to work

      IF(vSave_Frames)
         sprintf strFrame_Number, "%08d", iDex      //Using %08d so we (hopefully) don't have to worry about running out of digits
         SavePICT/O/E=-5/B=144 as strFrame_Path+strFrame_Base_Name+strFrame_Number+".png"
      ELSE
         Sleep/T 2   //If not making the frames, pause execution briefly
      ENDIF    //End checking to see if we are making frames or not
   ENDFOR      //End looping over the points in the wave

END
```

Sample code for making the frames

When creating the frames, you want things that are the same in every frame (axes, axis labels, legends, color scales, etc) to not move. If they move between frames then it will look bad when you create the video. The easiest way to get around this is to make sure you provide fixed numbers for any setting that can also take Auto (such as the graph and margin sizes). If you have something that does change, such as a time, you might want to use a monospaced font such as Consolas.


Margins:
Left: Auto
Bottom: Auto
Right: Auto
Top: Auto
Width mode: Auto
Height mode: Auto

Don't use Auto for any of these parameters in Modify Graph

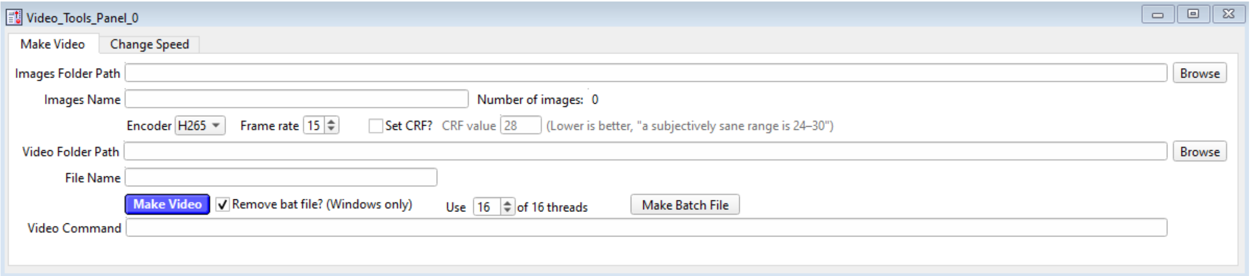
The ipf has some commands that assume that your images are pngs, so unless you want to get into the code I would stick with pngs. The video that gets created is an mp4. `FFmpeg` can make almost any other format of video you want, but I hardcoded mp4 for convenience. PNG images can be transparent, but this does not translate into the video. I haven't found a way to make videos transparent. Jpg is a lossy format, so you should avoid using this for making the frames.

There cannot be anything else in the images folder except the images. I'm not sure why you would do this, but if you do, the code will probably run into some issues.

4 Making the video

4.1 Making the panel

Once you have made the frames, load the ipf and run the function `Video_Tools_Make_Panel` either from the command line or from the "Useful Tools" menu (Useful Tools→Video Tools→Make Panel). This creates a little GUI panel, as shown below.



Images Folder Path
Images Name
Number of images: 0
Encoder H265
Frame rate 15
☐ Set CRF? CRF value 28 (Lower is better, "a subjectively sane range is 24–30")
Video Folder Path
File Name
☑ Remove bat file? (Windows only)
Use 16 of 16 threads
Video Command

Panel for making the vidoes

4.2 Input parameters

Browse to the folder that contains the images ("Images Folder Path"). If your images have nicely formatted names, the "Images Name" box will fill in. Select the folder to hold the completed video ("Video Folder Path"), and a name for the video. The default framerate (15 fps) works well, though I've used lower ones to slow movies down. The file type of mp4 is hardcoded in the ipf, but can be changed if desired.

`FFmpeg` has a lot of input parameters that you can set to fine tune things, but for what I've used it for (short videos in Powerpoint) the only parameter that I've really needed to change is the Constant Rate Factor (CRF). The default CRF has worked well, but you can change it if the file size is large or the quality is low. The other thing you can change is the encoder (the code that converts the static images into movies). `FFmpeg` has access to lots of encoders, but the only ones the Igor code currently supports are H.264 and H.265. H.265 takes longer, but the file size is generally smaller for the same image quality as H.264. For example, I made a movie using 1780 frames, each of which was ∼1.7 MB, for a total size of 2.9 GB. With H.264 encoding and a CRF of 24 the video file size was ∼20 MB. With H.265 and a CRF of 28 the file size was ∼15 MB. The video quality looked essentially the same. H.265 is slower, but the speed difference really wasn't a huge issue, and I liked having smaller video files. For a very simple or small video H.264 might give a smaller file, but for anything complex H.265 is the way to go. I tried AV1, and it was super slow (orders of magnitude slower than H.265). This might have been a function of the hardware and version of `FFmpeg` that I was using, and this will probably change in the future.

There is an option to change the encoder speed, which can also improve the compression ratio at the expense of speed, but right now this is not set, so the default of medium is used. In my tests I didn't see much improvement with different preset speeds, and the difference in time was pretty substantial.

If you want to know more about the parameters mentioned above, then read the `FFmpeg` documentation. It's generally pretty good. You can also find lots of other resources online (e.g. StackExchange).

By default `FFmpeg` uses all the available threads on your computer. There is an option to specify the number of threads that get used in case you don't want to overstress your computer.

4.3 Calling `FFmpeg` with ExecuteScriptText

After you click the "Make Video" button, Igor will use a command called **ExecuteScriptText** to create the video. On Windows, the code first creates a batch file with the proper command and then uses **ExecuteScriptText** to execute the batch file. The batch file is in the same folder as the video and will be deleted after making the movie, though there is an option to keep the batch file if you run into issues. On a Mac Igor passes the command directly to AppleScript (no batch file), which is why the commands are a little different. If everything was entered correctly, `FFmpeg` should run and create a nice video. The command that was executed will show up in the box below the "Make Video" button if you run into issues and want to call the commands from the terminal prompt.

4.4 Exif data

Information about the video will get stored in the Comment section of the video's Exif data. You can use `FFmpeg` to read this data or a separate (and free and cross-platform) program called ExifTool. ExifTool can also be called from Igor using **ExecuteScriptText** and the appropriate commands. The experiment that was used to create the movie as well as the path to the image source folder will be stored in the Exif data in case you lose track of how you made the movie.

5 Changing the video speed

The second tab of the panel has controls for changing the speed of an already existing video. I've used

```
-filter:v "setpts=X*PTS"
```

for this, where "X" is the factor by which you want to slow down the movie. More details are given here: `https://trac.FFmpeg.org/wiki/How%20to%20speed%20up%20/%20slow%20down%20a%20video`. The source video file name and path as well as the amount the speed was changed will be stored in the Exif data of the new video.