# A guide on making animated .webp emotes for 7TV

## Introduction

As much as we all love to make animated emotes, GIF is a highly outdated format by modern standards. Supporting only 256 colors, limited framerate and no variable transparency, calling it an obsolete format for animated emotes would be an understatement. Luckily, newer formats have been developed as the time went on, such as WebP and AVIF. They allow features GIF majorly lacks, like control over transparency, full color palette, better compression and smaller file sizes. Since AVIF is still too niche and isn't as widely used yet, this guide will focus on making animated emotes in .webp format. We're going to use Adobe Photoshop on Windows, but you're free to provide tutorials for other operating systems and software. Please note that this guide will NOT be teaching you how to animate in Photoshop, only how to RENDER an emote.

Here's how both GIF and WebP look on an emote with a variable transparency (courtesy of Shmovy):
GIF
WebP

And here's the example of a color palette:
GIF
WebP

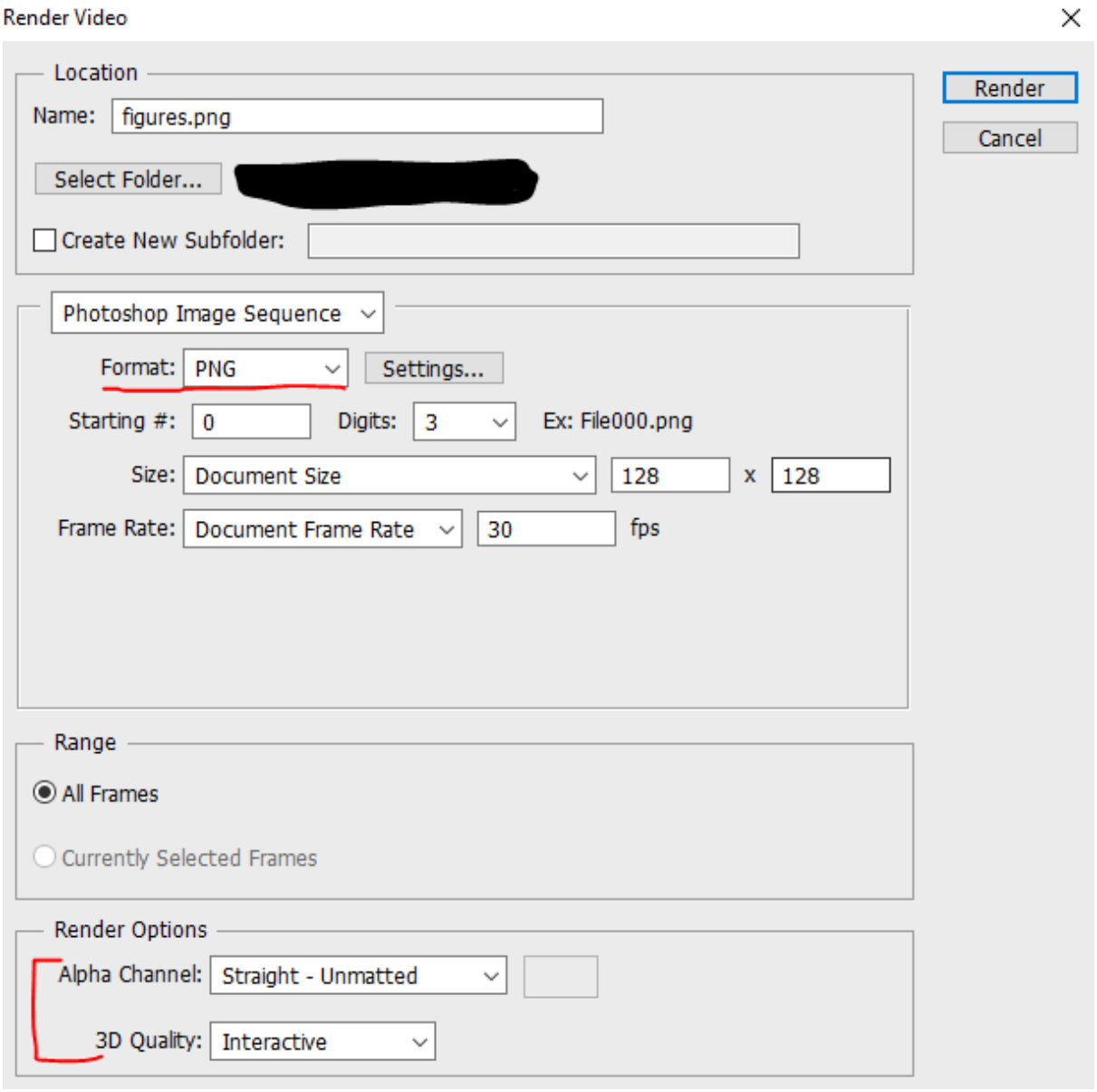
## Tools

- Image/video editing software of your choice. Make sure it supports exporting as image sequence.
- ffmpeg

## Step 1: Exporting your frames

Assuming you're already familiar with how the timeline in Photoshop works, you should have your animated emote prepared. Once you're done tweaking your

frames, go to File -> Export -> Render Video..., which should bring you to this window:




Let's go over each parameter.

Name - should be obvious, it's the name your frame files will have.

You have to select a folder to save your frames to. Do that using Select Folder.... You can also create a subfolder for the frames to make it more convenient for yourself.

Make sure Photoshop Image Sequence is selected in the drop-down menu.

Now, the first most important parameter, conveniently marked in red. In the Format drop-down menu, make sure PNG is selected.

In the Starting # leave the digit as 0. In the Digits, you select a number 0 through 5 depending on how many digits you want your frames to end with. For example, if you select 0, your frames will be named File0.png, with 3 it's File000.png, with 5 it's File00000.png... you get the idea. In this case, the digit is set to 3.

In the Frame Rate menu set your desired frame rate. Do note that 7TV currently caps at 50 fps. Also, the 7tv processor only uses gif timings, so it's limited to 10 ms increments. Here are all the possible output timings:

10, 12.5, 16.666, 20, 25, 33.333, 50

Now comes another important part: Render Options.

Alpha Channel must be set to Straight - Unmatted. If set to anything else, you'd either lose your alpha channel or your emote would have ugly black/white border, which you don't want.

Leave 3D Quality as Interactive.

Once you're set, click Render and wait for your frames to render. Depending on how many frames your emote has and how fast your computer is, it may take from 5 to 20 minutes, so keep that in mind.

To practice, you'll be provided with a simple .psd project file: basic geometric shapes appearing and disappearing. It's meant to showcase the fact that .webp supports alpha channel and partial transparency .gif lacks. Get it down below:

External link

# Step 2: ffmpeg

There's a good chance you already *have* ffmpeg installed on your PC. You can easily check it by opening the command prompt and typing in ffmpeg. If ffmpeg is installed, a similar menu should appear:

```
ffmpeg version 4.3.2-2021-02-02-full_build-www.gyan.dev Copyright (c) 2000-2021 the FFmpeg developers
  built with gcc 10.2.0 (Rev6, Built by MSYS2 project)
  configuration: --enable-gpl --enable-version3 --enable-static --disable-w32threads --disable-autodetect --enable-fontc
onfig --enable-iconv --enable-gnutls --enable-libxml2 --enable-gmp --enable-lzma --enable-libsnappy --enable-zlib --enab
le-libsrt --enable-libssh --enable-libzmq --enable-avisynth --enable-libbluray --enable-libcaca --enable-sdl2 --enable-l
ibdav1d --enable-libzvbi --enable-librav1e --enable-libwebp --enable-libx264 --enable-libx265 --enable-libxvid --enable-
libaom --enable-libopenjpeg --enable-libvpx --enable-libass --enable-frei0r --enable-libfreetype --enable-libfribidi --e
nable-libvidstab --enable-libvmaf --enable-libzimg --enable-amf --enable-cuda-llvm --enable-cuvid --enable-ffnvcodec --e
nable-nvdec --enable-nvenc --enable-d3d11va --enable-dxva2 --enable-libmfx --enable-libcdio --enable-libgme --enable-lib
modplug --enable-libopenmpt --enable-libopencore-amrwb --enable-libmp3lame --enable-libshine --enable-libtheora --enable
-libtwolame --enable-libvo-amrwbenc --enable-libilbc --enable-libgsm --enable-libopencore-amrnb --enable-libopus --enabl
e-libspeex --enable-libvorbis --enable-ladspa --enable-libbs2b --enable-libflite --enable-libmysofa --enable-librubberba
nd --enable-libsoxr --enable-chromaprint
  libavutil      56. 51.100 / 56. 51.100
  libavcodec     58. 91.100 / 58. 91.100
  libavformat    58. 45.100 / 58. 45.100
  libavdevice    58. 10.100 / 58. 10.100
  libavfilter     7. 85.100 /  7. 85.100
  libswscale      5.  7.100 /  5.  7.100
  libswresample   3.  7.100 /  3.  7.100
  libpostproc    55.  7.100 / 55.  7.100
Hyper fast Audio and Video encoder
usage: ffmpeg [options] [[infile options] -i infile]... {[outfile options] outfile}...

Use -h to get full help or, even better, run 'man ffmpeg'
```

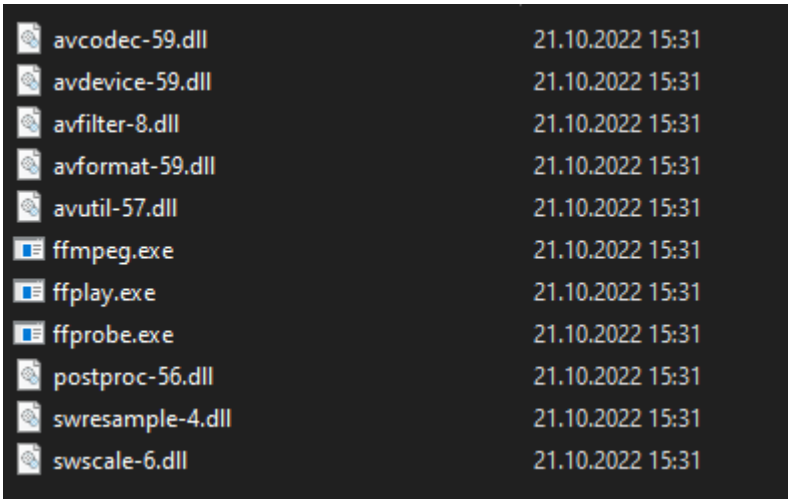
# Setting up ffmpeg

This part is relevant only if you don't have ffmpeg on your computer. If you do, you may skip this.

There exist many different builds of ffmpeg since it's an open-source project. We are going to use the build from BtbN: https://github.com/BtbN/FFmpeg-Builds/releases

Assuming you're on Windows, download ffmpeg-master-latest-win64-gpl-shared.zip and extract its contents somewhere on your computer.

Navigate to the bin folder and you should be presented with the following:




We're interested in ffmpeg.exe

### Adding ffmpeg to the PATH variable

In Windows, there exists a variable called PATH. In order to use ffmpeg from everywhere, we must add the exe to it.

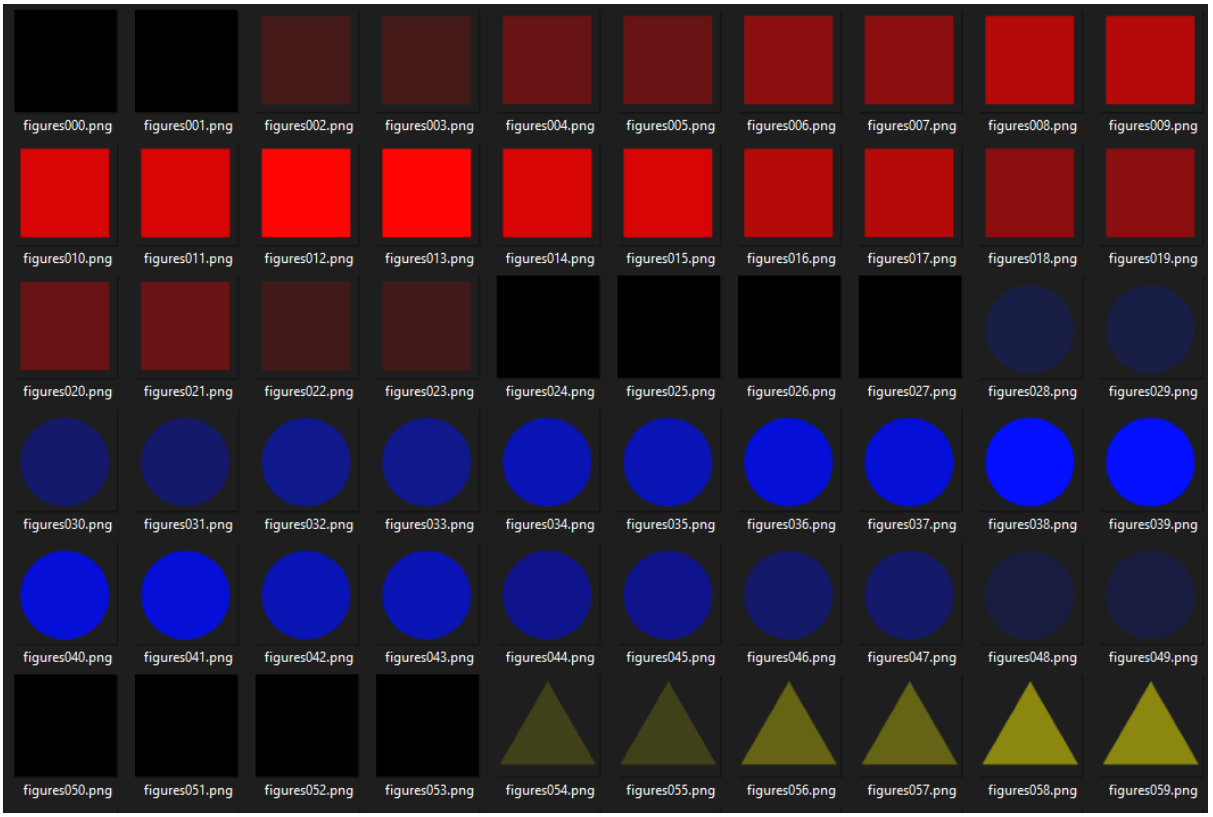
First, open up Windows' search bar and type in *"Edit the system environment variables"*. A window titled *"System Properties"* should appear. In the bottom-right corner, click on the Environment Variables... button. Find the *"Path"* variable and click on Edit... A new window should pop up. Now click on New, copy and paste the destination to the folder with the ffmpeg.exe file. It should look something like this:

```
J:\ffmpeg\bin
```

Save and close. If done correctly, next time you type ffmpeg in the command prompt, ffmpeg's menu should appear, similar to the one shown earlier. If it still doesn't do anything, retrace your steps and try again.

## Step 3: Rendering with ffmpeg

Let's go back to the folder you've exported your frames from Photoshop into. If you used the .psd file provided in the guide, the contents should look like this:




You can see that each individual frame has been made into a .png image. Now here comes the fun part: actually *animating* these images.

First, we must create a .bat script. Create a .txt file, open your favorite text editor and paste the following string:
ffmpeg -r %2 -start_number 0 -i "%1%%0%3d.png" -compression_level 6 -lossless 1 -quality 90 -loop 0 -y "%1.webp"
Save it in the same folder your ffmpeg.exe is in. Make sure that the file has .bat extension. If done correctly, the script should be universal for each emote you'd make. Open up PowerShell (press Shift+right-click), then type in the following: <script>.bat <sequence prefix> <emote fps> (keep in mind that 7TV caps at 50 fps) <how many digits long your total amount of frames>

In our case, it would be: <script>.bat figures 30 3

If done correctly, a .webp file should appear in the same folder. Here's what the end result looks like:

External link

If you didn't get this file to render, retrace your steps and try again.

Congratulations, you've had enough patience to finish this tutorial! Now get out there and start making emotes!

# Extras: Explaining ffmpeg parameters

In case you wish to learn more about the parameters we've used:

Most of the time, you'll want to encode your webp losslessly, so you'd specify the -lossless 1 argument.

If the resulting file is too big, you can remove that argument to encode a lossy webp, although the quality will be lower.

You can tune the following settings to make lossy encodes look better, but still keep it small enough to upload.

-compression_level goes from 0 to 6, where 4 is the default. In lossy mode, higher numbers are slower but yield better looking compression. In lossless mode, higher numbers mean smaller files, but longer encodes.

-quality goes from 0 to 100, where 75 is the default. In lossy mode, it's similar to a "bitrate" control, where higher numbers mean larger files, and better quality. In lossless mode, where higher numbers mean smaller files, but longer encodes.

If your lossless encode is just slightly too large to upload, you could try using higher -compression_level and -quality values to get it under the threshold. Using -quality 100 in lossless mode, or -compression_level 6 in either mode will make the encode VERY slow, so be warned.

# Credits:

I'd like to thank the following people for making this guide happen:

-AnsonX10: for introducing me to rendering in .webp, explaining what the parameters do;
-hyruverse and Melonify: for coming up with a batch script and making the guide more compact;
-People who made ffmpeg a thing.