How to use OBS video output for video conferences on linux

David Mazières

December, 2020

Introduction

OBS is a real-time video editor, letting you combine multiple video sources. In particular, you can combine a camera source with the capture of an application window, such as a PDF viewer showing slides. OBS is particularly effective using a "chroma key" (a.k.a. green screen) effect. You can sandwich your slides between a camera image of yourself and a virtual background. This allows you to present slides to your audience the way a weatherman shows the weather, pointing to content on your slides as if they were projected behind you in a lecture hall. If you use a green screen, any slides or whiteboard you show behind yourself will be much more legible than if they were actually physically behind you. As a bonus, the green screen lets you give a normal-looking lecture even if you don't have a projector, large television, or whiteboard available.

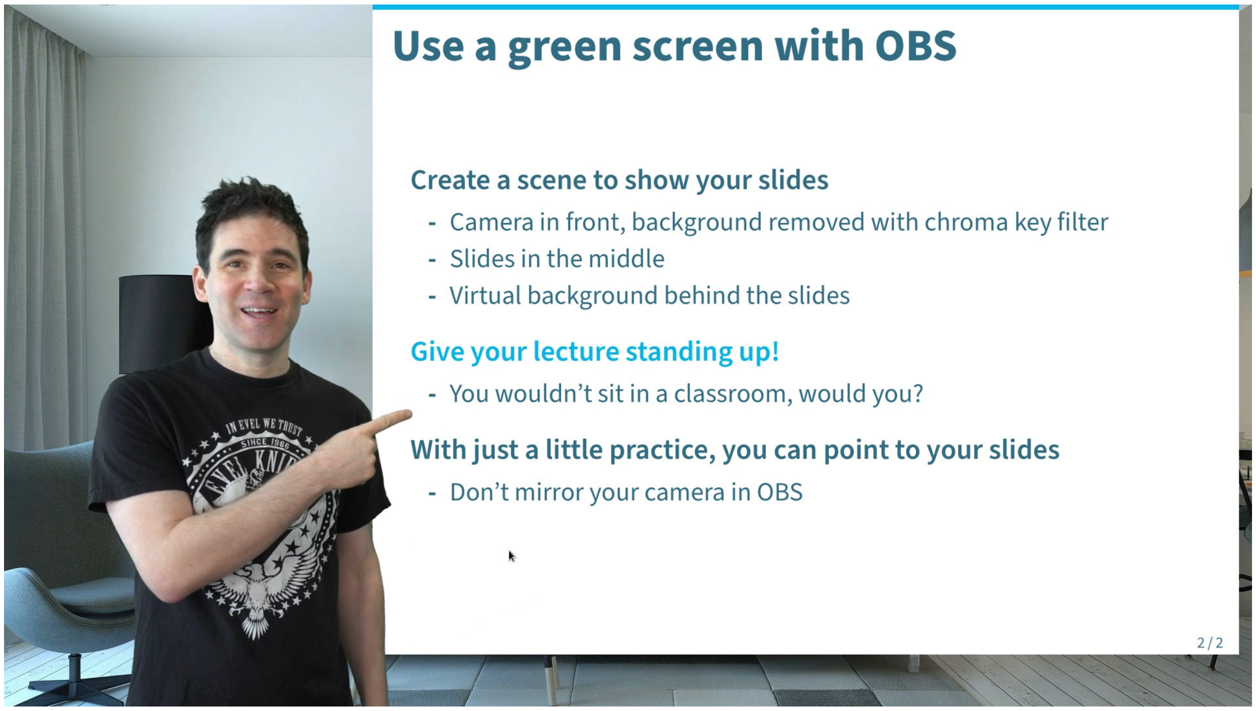
This blog post is about how to make OBS work for videoconferencing, particularly when you are presenting using in-browser video-conference systems such as jitsi, Google meet, Google duo, and talky. There are two parts. First, you need to get OBS outputting video to a virtual camera device. Second, you need to hide the real camera devices so that chromium doesn't try to select the wrong video device.

I favor arch linux, so will describe the exact steps required on arch, but most of the tricks will work on other linux distributions. I use the yay AUR helper to install packages.

Setting up v4l2loopback to work with OBS

The first thing to do is to install a video4linux 2 "loopback" device that you can use as a virtual camera. To do this, you will need to install the v4l2loopback driver and the obs-v4l2sink plugin for OBS. It's possible that as of obs 26.1 you can get by without obs-v4l2sink using the new virtual camera interface, but I'm still using the plugin because it's nice to be able to set it to be on by default. As of this writing, arch is still on OBS 26.0, but I can verify that obs-v4l2sink still works in 26.1.

On arch, the command to install the needed software is:


Use a green screen with OBS

Create a scene to show your slides
- Camera in front, background removed with chroma key filter
- Slides in the middle
- Virtual background behind the slides

Give your lecture standing up!
- You wouldn't sit in a classroom, would you?

With just a little practice, you can point to your slides
- Don't mirror your camera in OBS

2 / 2

Figure 1: Background courtesy of pixabay

```
yay -S v4l2loopback-dkms obs-v4l2sink-git
```

To load the v4l2loopback device by default and also immediately, you will need to run the following commands as root:

```
echo options v4l2loopback nr_devices=2 exclusive_caps=1,1,1,1,1,1,1,1 video_nr=0,1

echo v4l2loopback > /etc/modules-load.d/v4l2loopback.conf

modprobe v4l2loopback
```

Some important things to note. First, we are creating only two devices, but `exclusive_caps` contains values for 8 devices. This is actually necessary—for some reason it does not work if you have fewer entries in `exclusive_caps`. It will *seem* to work, but various video conference systems will refuse to use your virtual camera. Second, note that we are naming the devices `v4l2lo0` and `v4l2lo1`. These names will become important later on when we write udev rules to hide your non-virtual cameras. Finally, we are assigning video device numbers 0 and 1 so that our virtual cameras show up first.

Next, you need to configure OBS to output to one of your virtual cameras. Having installed the `obs-v4l2sink-git` plugin, if you restart OBS and look in the tools menu, you should see a "V4L2 Video Output" option, which will give you a dialog you can configure to "auto start" as well as to start immediately as follows:

Above I told you how to configure the v4l2loopback driver to have two loopback devices, `/dev/video0` and `/dev/video1`, so choose one of these. There's no way to close the dialog,

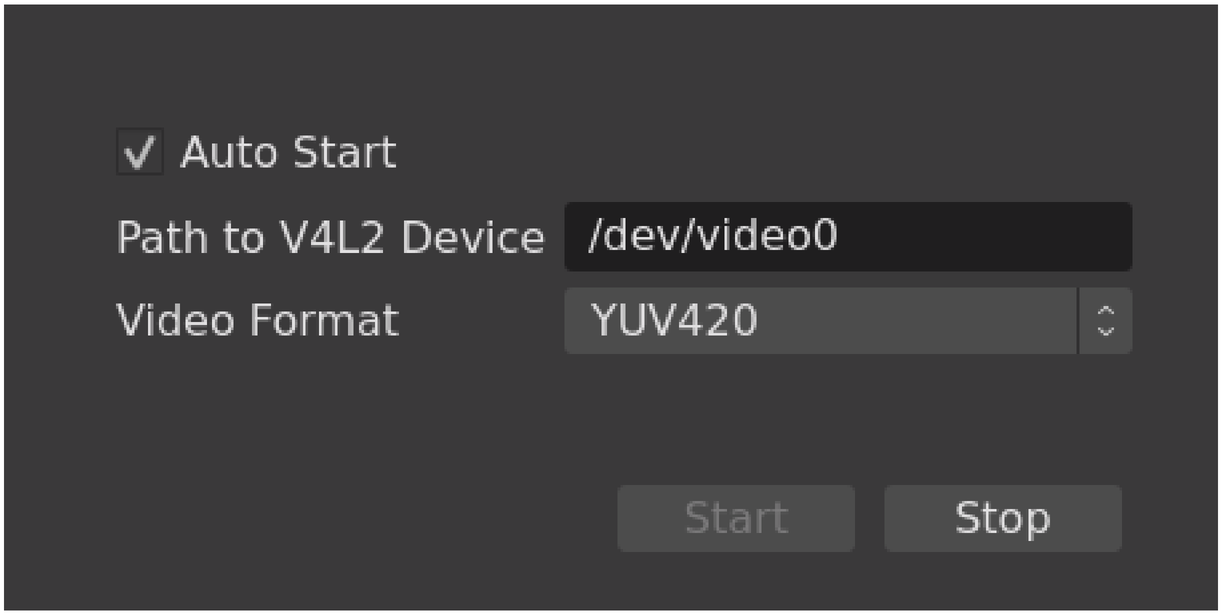

☑ Auto Start
Path to V4L2 Device: /dev/video0
Video Format: YUV420

Figure 2: V4L2 Video Output dialog

so just send it a Quit message (e.g., Alt-F4 with many window managers). Note that once you have started V4L2 output, certain options in the OBS settings menu will be locked. If you need to change these settings, just re-open the “V4L2 Video Output” dialog and click “Stop”, go back into the settings menu to make your adjustments, and finally click “Start” to resume your v4l2loopback feed.

At this point, you should test your configuration. Configure a physical camera as a V4L2 video source (it will be useful to have your real camera working before the next part). In the settings for your physical camera, make sure you de-select the “Use Buffering” option, or you will get annoying delay in your video conference. Finally, before moving on, test your virtual camera using, for example:

```
    vlc v4l2:///dev/video0
```

Getting chromium to open the virtual camera

The instructions above may be good enough to get OBS working with zoom, which allows you to select a camera. Unfortunately, they don’t work reliably with web sites. Often, regardless of the camera you select in the chromium settings, when a web site attempts to use your camera, chromium will attempt to use your *real* camera rather than the virtual one. This then fails because OBS already has your real camera open, so the web site will end up concluding that it must not have camera access on your system.

Frustratingly, if you click the small camera icon in the address bar, it will *look* like chromium is offering you a choice of video devices, but it won’t *actually* let you choose a device. What appears like a drop-down menu does absolutely nothing when you click on it. Perhaps there’s some catch-22 in which the web site has requested a particular camera and when this fails the web site has restricted you from asking the browser for another camera.

A solution is to hide your real cameras from most applications including chromium, so that your virtual camera is the only camera they see. Before you do this, make sure you have added your real camera (likely `/dev/video2` when 0 and 1 are taken by `v4l2loopback`) as a V4L2 source in a scene for OBS. Also make sure you have named your loopback devicves `v4l2lo[0-1]` as described above, or you will need to adjust the instructions accordingly. Finally, quit OBS and run the following commands as root:

```
cat > /etc/udev/rules.d/99-hide-cam.rules <<'EOF'

SUBSYSTEM!="video4linux", GOTO="hide_cam_end"
ATTR{name}=="v4l2lo[0-9]", GOTO="hide_cam_end"

ACTION=="add", RUN+="/bin/mkdir -p /dev/obs-only"
ACTION=="add", RUN+="/bin/mv -f $env{DEVNAME} /dev/obs-only/"
ACTION=="add", ATTR{index}=="0", RUN+="/bin/ln -fs $name /dev/obs-only/$env{ID_SERI

ACTION=="remove", RUN+="/bin/rm -f /dev/obs-only/$name"
ACTION=="remove", RUN+="/bin/rm -f /dev/obs-only/$env{ID_SERIAL}"

LABEL="hide_cam_end"
EOF

udevadm control -R

udevadm trigger -c add
```

These commands create a file `/etc/udev/rules.d/99-hide-cam.rules` that instructs udev to hide any camera devices that aren't named `v4l2lo[0-9]` by moving them to a directory called `/dev/obs-only` where applications won't look for them. It also creates symbolic names based on the device serial numbers, so that you can configure OBS to use a particular camera regardless of which device number that camera is assigned. There's one limitation, which is that if you use the udev seat tag to give console users access to camera devices, then this won't work if your camera is plugged in before you log in. A workaround is just to add yourself to the `video` group.

Like most applications, OBS won't look for cameras in the `/dev/obs-only` directory. Moreover, it also won't let you add such cameras from the UI. But since you stopped OBS before running the above commands, you can just edit the configuration file manually before restarting it. Go to the directory `~/.config/obs-studio/basic/scenes`, and edit each of the JSON files there. Either search for the string "`/dev/video`" and replace it with a new path starting `/dev/obs-only/`, or search in the `sources` array for entries with `"id": "v4l2_input"`, and change `settings.device_id` to the appropriate path under `/dev/obs-only`. Use the name with a serial number if you plan to have more than one physical camera plugged into your computer. For example, your JSON file might contain something like this (where the line with `device_id` has been edited):

```
{
    "balance": 0.5,
    "deinterlace_field_order": 0,
    "deinterlace_mode": 0,
    "enabled": true,
    "flags": 0,
    "hotkeys": {},
    "id": "v4l2_input",
    "mixers": 0,
    "monitoring_type": 0,
    "muted": false,
    "name": "Video Capture Device (V4L2) 2",
    "prev_ver": 436207618,
    "private_settings": {},
    "push-to-mute": false,
    "push-to-mute-delay": 0,
    "push-to-talk": false,
    "push-to-talk-delay": 0,
    "settings": {
        "device_id": "/dev/obs-only/046d_Logitech_Webcam_C930e_4192C0EA",
        "input": 0,
        "pixelformat": 861030210
    },
    "sync": 0,
    "versioned_id": "v4l2_input",
    "volume": 1.0
}
```

Screen-sharing your virtual camera

Most videoconference systems allow you to share your screen or a window on your screen, in addition to providing a camera feed. Though typically only one participant can share a screen at a time, if you are teaching a class or giving a presentation, you should have the option to share your screen. Most video conference systems treat a shared screen differently from a camera feed. Under ideal conditions, the two might be identical. However, if a recipient has low bandwidth, generally the videoconference system will compensate by reducing the resolution of a camera feed, while instead reducing the frame rate of a screen share. Also, when there's a mismatch of aspect ratio, a screen share will typically be scaled to show the full screen to each participant, while camera feeds are sometimes cropped, potentially clipping content from your slides.

You can use your OBS output as a screen share instead of or in addition to as a virtual camera. To do this, click anywhere outside a source to make sure no source is selected, then right-click on the Program (or Preview if you are not using studio mode) portion of your OBS window to select "Windowed Projector (Program)" (or "Windowed Projector (Preview)").

This will bring up a small window with your OBS video feed. Resize the window to 1920x1080, either through a hotkey in your window manager or by running a command such as `xdotool search --name "Windowed Projector" windowsize 1920 1080`. Now you can share this preview window using your videoconference system and participants will get your OBS output as a screen share.

You might wonder which is better, a screen share or a virtual camera. Unfortunately, different participants appear to have different preferences. In a survey of students in the class I taught last Spring (when the COVID19 pandemic forced everyone to move from in-person instruction to zoom):

- 27% of students preferred watching me as a screen share
- 13% preferred watching me as a normal camera view
- 47% wanted both views available
- 13% said either view was fine and they only needed one

So based on this feedback, I now choose to provide both feeds, even though it is redundant. Obviously this works only because my own internet uplink speed is very fast and I can upload two full HD video feeds without any problem. My concern is serving students who might not have good internet connectivity, rather than conserving my own bandwidth.

Sharing an arbitrary application

Sometimes you want to share a real application window (not an OBS projector window) with someone over OBS. You do this by adding a Window Capture (Xcomposite) source to an OBS scene. Likely you will have several standard scenes for the windows you share the most, such as your PDF viewer for slides and your whiteboard application for a whiteboard (I use openboard). But what about when you quickly want to share an arbitrary other window?

Most Xorg applications have an option `-name` or `--name` to specify the name by which the application should be listed in X resources and in the initial window manager title bar. You can create a scene that generically shares the "OBS" window scaled to inner bounds of 1920x1080. Then if you want to show a demo, you can run `xterm -name OBS`. If you want to show code to someone, you can run `emacs -name OBS`. Or you can run `gimp --name=OBS`, `inkscape --name=OBS`, `wireshark -name OBS`, etc., and you don't have to worry about OBS selecting the wrong window that you'll have to manually reconfigure in the middle of a class or videoconference.

Zoom-specific issues

There are a couple of zoom-specific issues that are quite annoying with screen sharing. First, you probably want to enable "Use dual monitors" in the "General" panel of zoom settings, so that you can still see other people even when sharing a window. Otherwise, you can't see the people in your meeting.

An annoying problem is that zoom doesn't seem to use the XComposite extension, with the result that if a window you are sharing gets at all obscured by another window, the covered portion turns black. Note that this even includes the portions of the shared window that are covered by zoom's own toolbar!

To work around this problem, it's useful to create a "fake" monitor onto which to move your OBS projector window (since you don't need it taking up precious screen real estate—you've already got the main OBS window). A fake monitor is a real output of your video card to which there is no monitor connected (though with x11vnc you can potentially use a tablet or laptop VNC client to access the contents of the fake monitor).

If you have an NVIDIA video card and are using their Xorg driver, you can create a fake monitor by means of the `ConnectedMonitor` directive in the `Device` section of your `xorg.conf` file, which tells the driver to pretend there is a monitor connected to your output even if none is detected. Note that the documentation implies you should be able to place this in the `Device` or `Screen` section, but I learned the hard way that this does not work in the `Screen` section. Here's an excerpt from a working `xorg.conf` file with two real 2560x1440 monitors above two fake 1920x1080 monitors:

```
Section "Device"
    Identifier     "Device0"
    Driver         "nvidia"
    VendorName     "NVIDIA Corporation"
    BoardName      "Quadro M5000"
    BusID          "PCI:33:0:0"
    Option         "ConnectedMonitor" "DP-0,DP-4,DP-7,DP-2"
EndSection

Section "Screen"
    Identifier     "Screen0"
    Device         "Device0"
    Monitor        "Monitor0"
    Option         "nvidiaXineramaInfoOrder" "DP-0,DP-4,DP-7,DP-2"
    DefaultDepth    24
    Option         "metamodes" "DP-0: 2560x1440+0+0, DP-4: 2560x1440+2560+0, DP-7:
    SubSection     "Display"
        Depth       24
    EndSubSection
EndSection
```

DisplayPort allows two devices per physical port on your video card. However, the driver doesn't seem to allow you to use more than one of these, even with the `ConnectedMonitor` directive. Hence, make sure that the value $\lfloor n/2 \rfloor$ is unique for each device DP-n you list in `ConnectedMonitor` as well as any real monitors you didn't bother listing in `ConnectedMonitor`. Use the `xrandr` command (with no arguments) to see the names of your existing monitors—they probably don't correspond to anything sane like the physical position of the DisplayPort connectors on your video card.

If you have intel video instead of NVIDIA, you may be able to configure a fake monitor more simply by use of the `xrandr` command. There's a discussion about doing so for the related purpose of using tablets with VNC clients as secondary monitors on the archlinux message board. I don't have firsthand experience with AMD graphics cards, but the same technique might work there as well.

I use the fvwm window manager, and bind hotkeys to maximize a window on my fourth (fake) monitor, where I can move by OBS projector window, as well as to move things back from inaccessible screens to the current screen.

A final important point: when you share a screen or window, zoom gets rid of its main window and moves the zoom toolbar (including, importantly, the ability to stop sharing) into an *unmanaged* window, which doesn't play nicely with window managers. Specifically, if this toolbar gets sent to the wrong monitor (like one of your fake displays), there is no way to drag it to the correct monitor. To get myself out bad situations where I can't access the zoom toolbar and so can't even stop sharing a window, I have an alias for the following command, which brings the toolbar to my first monitor at least temporarily (as soon as you interact with it, it will disappear again, but if you stop the screen share then the toolbar will get reintegrated into a normal managed zoom window):

```
xdotool search --name as_toolbar windowmove 0 0
```

Conclusion

Once you have OBS working well on linux, there's no reason for other applications to be directly accessing your camera's v4l2 device. Pretty much the only other application I run directly on my camera is guvcview for bypassing OBS to debug problems or for accessing raw camera controls. Even after the suggestions in the blog post, you can still access your camera (when OBS is not running) with `guvcview -d /dev/obs-only/video2`. If you want to change camera controls such as pan and zoom while OBS is open, use the `-z` flag to avoid competing with OBS for the video feed: `guvcview -z -d /dev/obs-only/video2`. Note that OBS already provides access to most camera controls in the camera source properties, but when you have issues and want to know if they are caused by OBS or your camera, guvcview can be handy.

With OBS output as your standard camera, you now have a large number of options that work no matter what platform you are using. For instance, maybe you usually lecture over zoom, but are now talking to someone using google duo, or were invited to give a presentation over webex. If you want to bring up your whiteboard to illustrate something, or share your text editor to take notes together, or bring up your slides and point to them, it's the exact same OBS hotkeys you've programmed regardless of the videoconferencing system in use. The uniformity of interface ensures consistently seamless transitions with minimal mental effort, which is the closest we are going to get to in-person meetings until the COVID19 pandemic is behind us.