Editing Video with Code

Using GStreamer Editing Services

James Henstridge <james@jamesh.id.au>
Perth Linux Users Group, May 2021

Background

At the start of the pandemic, PLUG needed a system that would let us continue to run meetings online.

Big Blue Button was the chosen platform:

- Open Source
- No client app or login required: works with standard web browsers
- Support for recording
- An install was available, maintained by UCC

The Problem

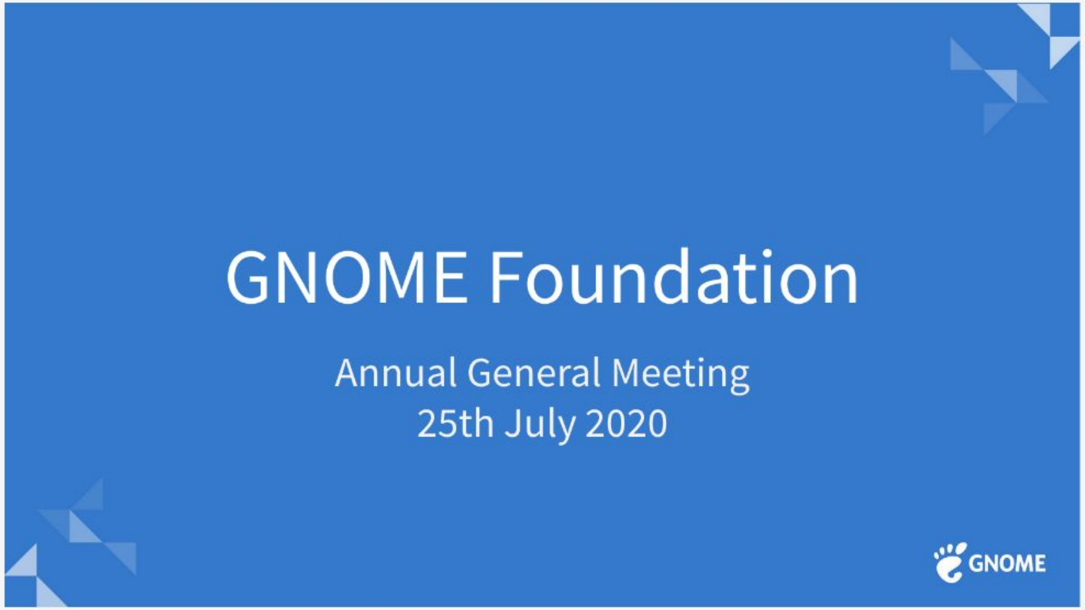
Big Blue Button recordings can only be viewed through BBB itself

If we stopped using BBB or switched instances, would we lose our recorded talks?

Older talks were published as videos on the website and YouTube. Can we convert BBB recordings into standard videos too?

Ideally something that looks a bit nicer than BBB’s own playback interface

GUADEC 2020 - TRACK 1



GNOME Foundation

Annual General Meeting
25th July 2020

Recorded with BigBlueButton.

BBB’s Playback System

Web page that uses JavaScript to coordinate

Video file for webcam capture + audio

Image files for slides

SVG for scribbles

Video for screen share footage (optional)

The Idea

Phase 1: scrape recording assets from BBB.

Phase 2: ?

Phase 3: ~~profit~~ render a video

First Attempt

GStreamer can probably do it, right?

“compositor” element can mix multiple video sources together

Set up timers to reconfigure the elements on slide changes, etc

Add encoder elements to the pipeline to output a video file

Problems

Simple use of timers relies on the pipeline running in real time (slow)

Need to make sure pipeline doesn’t stall or crash when reconfiguring

Is there a better way?

GStreamer Editing Services

Built on top of GStreamer

A toolkit for building non-linear video editing software

The core of the Pitivi video editor

Is a GObject based API, so easy to use from languages with gobject-introspection bindings (e.g. Python, JavaScript, etc)

Non-linear video editing

Also known as non-destructive video editing

Rather than modifying the source video files, projects consist of a list of instructions used to assemble the final video (sometimes known as an “edit decision list”)

Makes it easy to reuse assets in multiple projects, or apply the same effects to different assets

GES Project Structure

A GES Project has:

- Assets: videos, sound files, images, etc.
- A timeline
- One or more layers
- Assets are added to layers as “clips”, possibly transformed by an effect
- Clips within a layer cannot overlap, but may include transitions
- Clips in different layers can overlap, and are rendered on top of each other

Initialising GES from Python

```
import gi
gi.require_version('Gst', '1.0')
gi.require_version('GES', '1.0')
from gi.repository import Gst, GES

Gst.init(None)
GES.init()
```

Creating the project and add layers

```
timeline = GES.Timeline.new_audio_video()
project = self.timeline.get_asset()

layer1 = timeline.append_layer()
layer2 = timeline.append_layer()
...
```

Adding clips to layers

```
asset = GES.UriClipAsset.request_sync('file:///...')
project.add_asset(asset)

start = 10 * Gst.SECOND     # when the clip should start playing
inpoint = 0                 # time offset within video asset
duration = 30 * Gst.SECOND  # how long the clip should be
clip = layer.add_asset(asset, start, inpoint, duration,
                       GES.TrackType.UNKNOWN)
```

What next?

We have a timeline full of clips. Now what do we do?

Option 1: Create a GES.Pipeline GStreamer element and hook it up for playback, or to encoder elements

Option 2: save the timeline to an XML project file, so it can be processed with other tools

Saving the project

```
timeline.commit_sync()
timeline.save_to_uri('file://.../project.xges', None, True)
```

Using Project files

Play back or encode project using ges-launch-1.0 program:

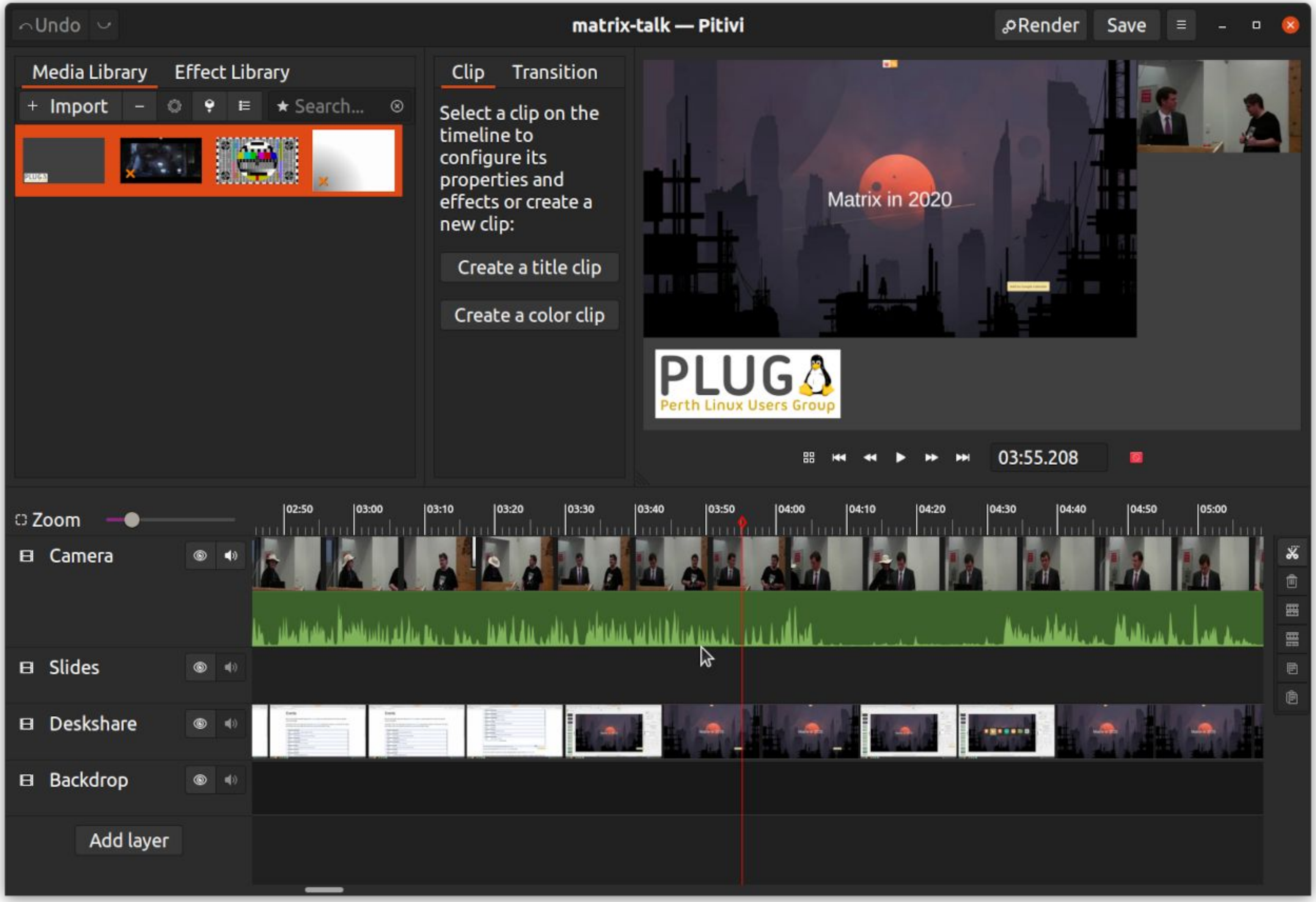
```
$ ges-launch-1.0 --load project.xges                # playback
$ ges-launch-1.0 --load project.xges -o project.mp4 # encode
```

Preview or edit in Pitivi video editor

Mapping a BBB recording to a GES project

Created a script that creates a GES project with:

1. Webcam video on one layer, scaled down and positioned in the corner
2. A layer for slides, with each image displayed using timings in the recording
3. Screen share on the next layer down, such that it is visible when no slide image is being displayed
4. A base layer showing a static image to fill the void and add branding



Select a clip on the timeline to configure its properties and effects or create a new clip:

Publishing on Github

As the scripts seemed like they might be generally useful, I published the git repository:

https://github.com/plugorgau/bbb-render

Followed up by a little promotion

Community responses

Used to publish videos for a number of conferences:

- GUADEC 2020 (GNOME)
- OpenAlt (Chech Republic)

Improvements from external contributors:

- Add start and end credits
- Support for BBB’s “whiteboard annotations” and red dot mouse cursor

When would you want to use GES?

If you have an edit list in some other format that you want to programmatically import into Pitivi

- e.g. use EXIF timestamps in photos to queue photos in a slide show

Create the same type of project many times

Can use it to start the project, finishing things off in Pitivi

Resources

bbb-render source code:

- https://github.com/plugorgau/bbb-render

Pitivi:

- http://www.pitivi.org/

GStreamer Editing Services documentation:

- https://gstreamer.freedesktop.org/documentation/gst-editing-services/