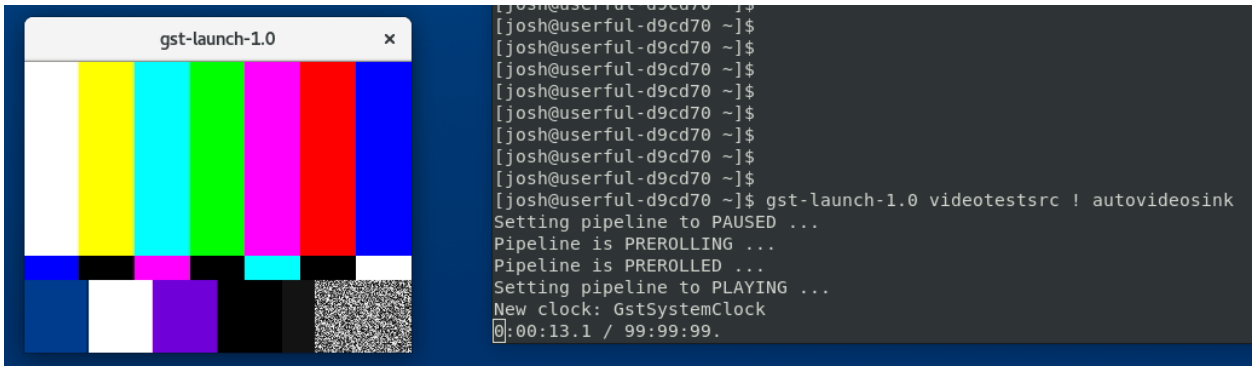
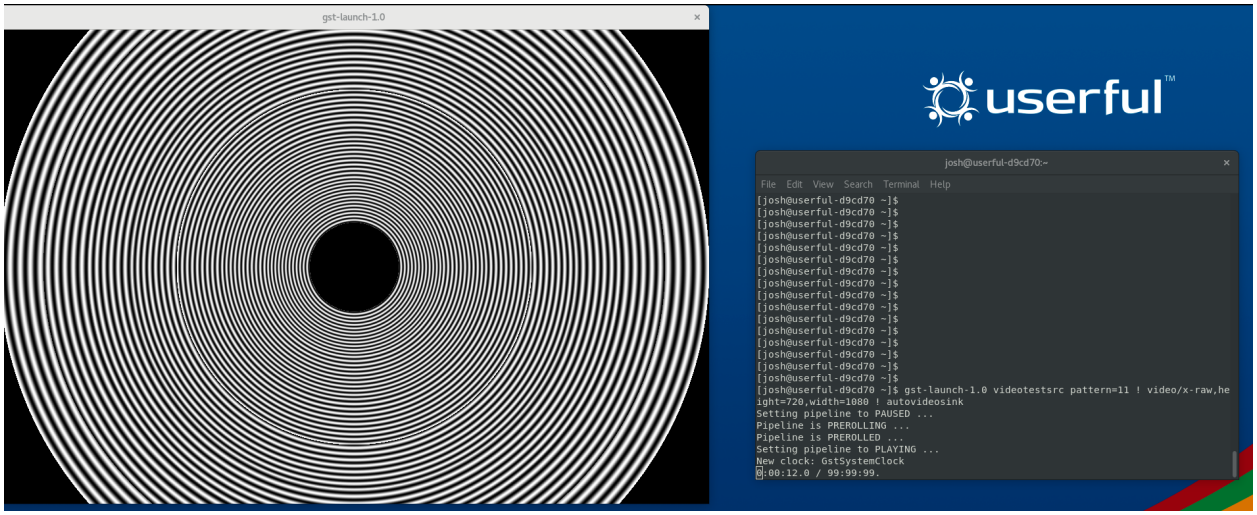
Gstreamer Workshop Exercises

Creating Simple Pipelines

Creation of simple pipelines via gst-launch-1.0, with selected properties options and some work with caps filters:





Using GStreamer Library for Python

Script that dynamically creates a pipeline for local media that can handle audio, video, or both using the main loop and other protocols as outlined in the workshop.

```python
import sys
import gi

gi.require_version('GLib', '2.0')
gi.require_version('GObject', '2.0')
gi.require_version('Gst', '1.0')

from gi.repository import Gst, GObject, GLib

def bus_call(bus, message, loop):
    t = message.type
    if t == Gst.MessageType.EOS:
        sys.stdout.write("End-of-stream\n")
        loop.quit()
    elif t == Gst.MessageType.ERROR:
        err, debug = message.parse_error()
        sys.stderr.write("Error: %s: %s\n" % (err, debug))
        loop.quit()
    return True
```

```python
def on_pad_added(element, pad, videosink):
    caps = pad.query_caps(None)
    name = caps.to_string()
    print("on_pad_added: ", name)
    print("element ", element.get_property("name"))

    if name.startswith('video'):
        video_sink = videosink.get_static_pad('sink')
        if video_sink and not video_sink.is_linked():
                pad.link(video_sink)
    elif name.startswith('audio'):

            convert = Gst.ElementFactory.make("audioconvert", "convert")
            resample = Gst.ElementFactory.make("audioresample", "resample")
            audiosink = Gst.ElementFactory.make("autoaudiosink", "audiosink")

            #pipeline.add(convert)
            audio_sink = convert.get_static_pad('sink')
```

```python
            if audio_sink and not audio_sink.is_linked():
                print("made it")

                #pad.link(audio_sink)
                on_pad_added.pipeline.add(convert)
                on_pad_added.pipeline.add(resample)
                on_pad_added.pipeline.add(audiosink)
                convert.link(resample)
                resample.link(audiosink)

                convert.set_state(Gst.State.PLAYING)
                resample.set_state(Gst.State.PLAYING)
                audiosink.set_state(Gst.State.PLAYING)

                pad.link(audio_sink)
```

```
58  def main(argv):
59      Gst.init(sys.argv)
60      loop = GLib.MainLoop()
61      #set debug level to errors
62      Gst.debug_set_default_threshold(3)
63
64
65
66      pipeline = Gst.parse_launch("filesrc location=sample_960x400_ocean_with_audio.wmv ! decodebin name=dec dec. \
67                                   ! autovideosink name=sink")
68      on_pad_added.pipeline = pipeline
69      sink = pipeline.get_by_name("sink")
70      decoder = pipeline.get_by_name("dec")
71      decoder.connect('pad-added', on_pad_added, sink)
72
73
74      bus = pipeline.get_bus()
75      bus.add_signal_watch()
76      bus.connect ("message", bus_call, loop)
77
78      ret = pipeline.set_state(Gst.State.PLAYING)
79      if ret == Gst.StateChangeReturn.FAILURE:
80          print("err2")
81          sys.exit(1)
82
83      try:
84          loop.run()
85      except:
86          pass
87      print ("shutting down pipeline")
88      pipeline.set_state(Gst.State.NULL)
89
90  if __name__ == '__main__':
91      sys.exit(main(sys.argv))
```

Twitch Streaming

By modifying the file twitch-linux.py to match the specifications of the local system, a Twitch stream was able to be started complete with screen recording and webcam video.

```
def main(argv):
    #initialising the GStreamer library
    Gst.init(sys.argv)
    Gst.debug_set_default_threshold(3)

    #This method is fast and useful when you don't want to handle connections between plugins manually and just want to launch some existing pipeline. Creating pi
    #This creates the pipeline but at first the pipeline is in NULL state (?)
    pipeline = Gst.parse_launch("v4l2src ! queue ! videoconvert ! queue ! video/x-raw, format=I420 ! vmix. "\
                                "ximagesrc ! queue ! videoconvert ! queue ! video/x-raw, format=I420 ! queue ! vmix. "\
                                "compositor name=vmix "\
                                "sink_0::xpos=0 sink_0::ypos=0 sink_0::width=300 sink_0::height=200 "\
                                "sink_1::xpos=0 sink_1::ypos=0 sink_1::width=852 sink_1::height=480 sink_0::zorder=1 sink_1::zorder=0 "\
                                "! queue ! video/x-raw, width=852, height=480, framerate=30/1, format=I420 ! "\
                                "x264enc tune=zerolatency bitrate=250 key-int-max=120 vbv-buf-capacity=2000 ! video/x-h264, profile=main "\
                                "! h264parse ! queue ! "\
                                #muxer.video "\
                                #"pulsesrc device=alsa_output.pci-0000_00_00.3.analog-stereo.monitor ! queue ! audioconvert ! audiorate ! audio/x-raw, rate=48000,
                                #"pulsesrc ! queue ! audioconvert ! audiorate ! queue ! audio/x-raw, rate=48000, channels=1 ! queue ! amix. "\
                                #"audiomixer name=amix latency=10000000  ! queue ! avenc_aac bitrate=64000 ! queue ! muxer.audio "\
                                #"flvmux
                                "muxer.video flvmux streamable=true name=muxer ! rtmp2sink name=rtmpstream")
```