DM n° 2 — Gestion de sous-titres dans un flux vidéo

Antoine de Roquemaurel (G1.1)

1 Partie I : en ligne de commande

1.1 Les formats OGV, Vorbis et Theora

Avant de pouvoir définir ces trois formats, il parait important de définir le projet OGG.

OGG est un projet libre issue de la fondation Xiph.org, ce projet à pour but de fournir des formats audio et vidéo qui soient libre. Plusieurs formats de fichiers ont ainsi vu le jour : `.oga` pour les fichiers audio, `.ogv` pour les fichiers vidéos et `.ogx` pour les applications.

Les formats que je vais définir ci-dessous sont tous issues du projet OGG.

OGV est le format issue du projet OGG permettant de lire des vidéos compressée tout en gardant une excellente qualité.

Vorbis est un format de compression/décompression audio libre, il peut faire concurrence au format propriétaire `.mp3`, cependant celui-ci est moins connu et n'est supporté que par peu de lecteur audio bien qu'il ait de meilleurs performance que son concurrent.

Theora est un format de compression/décompression vidéo libre, il est souvent couplé à *vorbis* pour l'audio, il est ainsi possible de lire une vidéo avec le son. Il est soutenu par de nombreux logiciels libres tel que Firefox ou les distributions GNU/Linux.

1.2 Lire une vidéo

1.2.1 Lire un fichier audio

```
gst-launch filesrc location=music.oga ! oggdemux ! vorbisdec ! audioconvert ! alsasink
```

Listing 1 – Lire un fichier audio – Uniquement pour les fichier `.oga`

```
gst-launch filesrc location=music.oga ! decodebin ! audioresample ! autoaudiosink
```

Listing 2 – Lire un fichier audio – Pour tous les formats audio

1.2.2 Lire une vidéo avec le son

```
gst-launch filesrc location=video.ogv ! oggdemux ! theoradec ! ffmpegcolorspace ! ↩
    autovideosink
```

Listing 3 – Lire une vidéo – Uniquement pour les fichier `.ogv`

```
gst-launch filesrc location=video.ogv ! decodebin ! autovideosink
```

Listing 4 – Lire une vidéo – Pour tous les formats audio-vidéos

Le plugin *decodebin* permet à *GStreamer* de détecter lui même la nature du conteneur et du codec à utiliser. Ainsi, nous pouvons lire une multitude de formats différents en utilisant celui-ci.
Dans le projet, seul *oggdemux* et *theoradec* ont été utilisés.

1.2.3 Lire une vidéo en remplaçant l'audio

```
gst-launch filesrc location=video.ogv ! decodebin ! autovideosink filesrc ↩
    location=music.oga ! decodebin ! autoaudiosink
```

Listing 5 – Lire une vidéo avec le son de source différente

1.3 Lire une vidéo avec des sous-titres

```
gst-launch filesrc location=video.ogv ! oggdemux name=demux \
  filesrc location=video.srt ! subparse ! overlay. \
  demux. ! queue ! vorbisdec ! audioconvert !  autoaudiosink \
  demux. ! queue ! theoradec ! ffmpegcolorspace ! \
  subtitleoverlay name=overlay ! autovideosink
```

Listing 6 – Lire une vidéo avec des sous-titres `.srt`

1.4 Enregistrer une vidéo avec les sous-titres incrustés

2 Partie II : en programmation C++

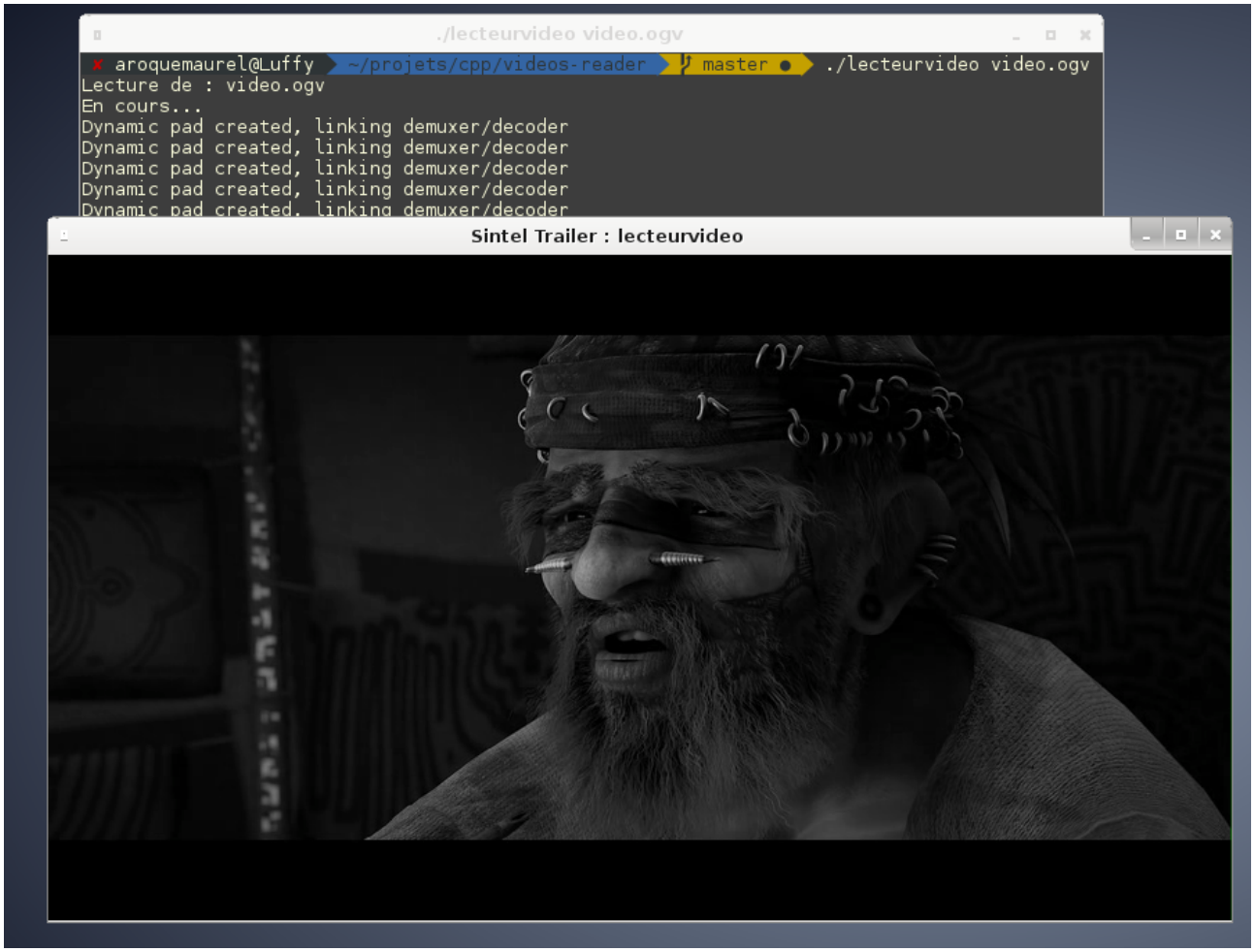
Pour cette partie, j'ai choisi de développer l'application en C++, comme proposé dans le sujet. En effet, j'affectionne tout particulièrement ce langage en raison de sa rapidité, de sa souplesse tout en restant extrêmement puissant et en utilisant le paradigme orienté objet.

L'application que j'ai développé ne permet de ne lire que des fichier au format `OGV` ou `Theora`, en raison des chaînes de traitement que j'ai choisi d'utiliser. Il aurait été éventuellement possible d'utiliser `decodebin` afin d'être ouvert au maximum de formats possible, cependant celui-ci rendait l'utilisation des sous-titres plus complexe.

Une autre solution permettant de mener ce projet à bien plus simplement aurait été l'utilisation de `playbin2`, cependant celui-ci n'étant pas autorisé pour réaliser ce sujet, je me suis restreint au format `OGV`.

2.1 Question 5

```
aroquemaurel@Luffy ~/projets/cpp/videos-reader <master> ./lecteurvideo video.ogv
```



Comme prévu dans le TP5, la vidéo est grisé est le son est réduit de 50%.

2.2 Affichage des sous-titres

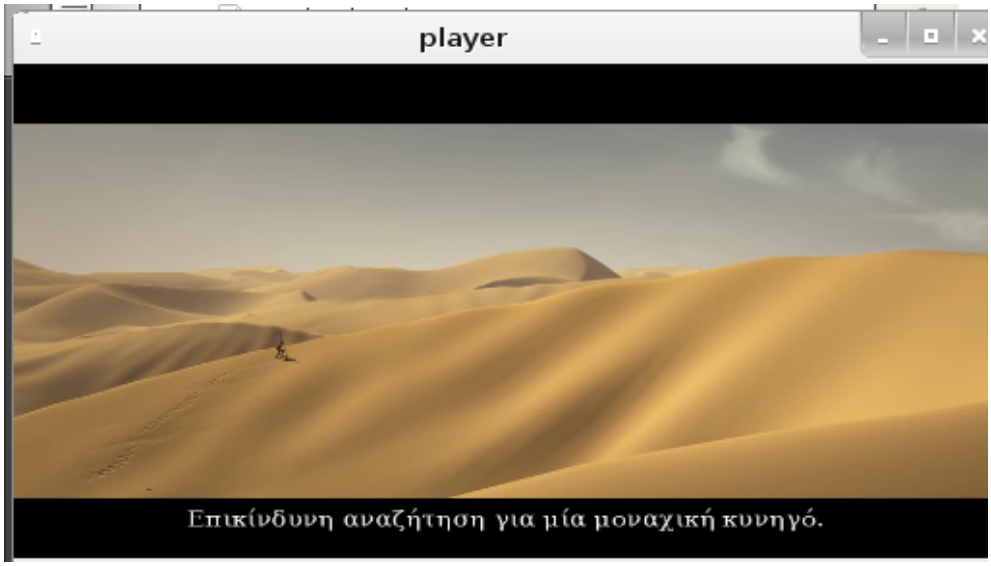
Afin d'afficher des sous-titres, j'ai utilisé la ligne de commande suivante :

```
gst-launch filesrc location=video.ogv ! oggdemux name=demux \
filesrc location=video.srt ! subparse ! overlay. \
demux. ! queue ! vorbisdec ! audioconvert ! autoaudiosink \
demux. ! queue ! theoradec ! ffmpegcolorspace ! subtitleoverlay name=overlay ! ↩
    autovideosink;
```

J'ai donc ajouté les éléments suivants, et ensuite effectué le linkage correctement :

```
_commands->addElement ("subOverlay",  "subtitleoverlay");
_commands->addElement ("subSource",   "filesrc");
_commands->addElement ("subParse",    "subparse");

_commands->setElement("subSource", "location", srtfilename);
```



2.3 Interface GTK

Mon projet à été pensé en respectant le pattern `M VC`, Modèle Vue-Contrôleur.

Modèle Le backend du logiciel, l'appel aux méthodes gstreamer.

Vue Ce qui a attrait à l'apparence et l'interaction, tout est présent dans la classe Ui

Contrôleur Qui se charge de faire le lien entre le modèle et la vue. Ici il est lié à la vue, on peut considérer que les méthodes de callback sont le contrôleur.

R GTK à été conçu pour le C et non pour le C++, ainsi cette bibliothèque fonctionne principalement avec un système de callbacks et de foncteur, chose difficilement réalisable dans une classe C++. J'ai donc une classe `Ui` possédant une majorité de méthodes statiques.
Il aurait été plus élégant d'utiliser la bibliothèque *Gtkmm*, bibliothèque GTK pensée pour le C++ permettant d'avoir toute la puissance du paradigme objet. Cependant, celle-ci n'étant pas installée en salle de TP, je n'ai pas pu m'en servir.

R Les images présents dans les boutons sont des images obtenus du gestionnaire de bureau, ici Mate, il se peut que chez vous ceux-ci s'affichent différemment, ou ne s'affiche pas du tout.
Cet affichage permet d'avoir un système d'exploitation uniformisé, GTK possède une multitude de boutons prédéfinis, pour cela il suffit de créer un bouton comme suit.

```
stop_button = gtk_button_new_from_stock (GTK_STOCK_MEDIA_STOP);
```

2.4 Fonctionnalités ajoutées au lecteur

Le lecteur précédent étant assez basique, plusieurs fonctionnalités ont été ajoutés au lecteur audio-vidéo afin de gagner en confort :

– Une barre « slider » permettant d'observer l'avancement du programme
– Un label affichant la durée de la vidéo, et le temps actuel parcouru. (en secondes)
– Une possibilité de mettre la vidéo en plein écran, à l'aide de la touche « F » ou « F11 ».
<++>

2.5

A Listings

B Table des figures