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

Antoine de Roquemaurel (G1.1)

1 Avant-propos

L'archive que vous avez obtenu sur Moodle contient plusieurs fichiers organisés comme suit :

`src/` Contient les sources C++ du projet, celles-ci sont détaillées section 1.1.

`player` Le binaire du lecteur vidéo, compilé chez moi.

`report_AntoinedeRoquemaurel.pdf` Le présent rapport que vous êtes en train de lire

`ressources/` Les vidéos et les sons utilisé en entrée d'une commande GStreamer. Celles-ci sont détaillées section 1.2.

1.1 Sources

Les sources contiennent un certain nombre de fichier :
- Les .h contiennent les classes développée pour le player
- Les .cpp sont les implémentations des précédentes classes
- Le makefile permet de compiler le projet, à l'aide de la commande `make`.

1.1.1 Détails des fichiers

Fichier `Makefile` Le fichier Makefile permettant de compiler le projet à l'aide de la commande `make`.

Fichier `main.cpp` Le point d'entrée de notre programme.

Classe `Ui` Contient la classe d'interface graphique à l'aide de GTK+.

Classe `Backend` Contient la classe permettant de faire le lien avec GStreamer.

Class `GStreamerCommand` Une classe permettant de faire une abstraction supplémentaire à l'utilisation de la bibliothèque GStreamer.

Classe `Player` La classe permettant de lancer le lecteur vidéo simplement.

1.2 Ressources

Voici les vidéos utilisées pour tester le lecteur vidéo ou des commandes GStreamer :

`video.ogv` La vidéo fournie

`video.srt` Les sous-titres de la vidéo fournie

`Thunderstruck.ogg` Un fichier audio permettant de tester les commandes GStreamer

`videoOutput.ogv` La vidéo de sortie suite à la question 4.

2 Partie I : en ligne de commande

2.1 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.

2.2 Lire une vidéo

2.2.1 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

2.2.2 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.

2.2.3 Lire une vidéo en remplaçant l'audio

```
gst-launch filesrc location=video.ogv ! decodebin ! autovideosink filesrc ↩
    location=Thunderstruck.ogg ! decodebin ! autoaudiosink
```

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

2.3 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`

2.4 Enregistrer une vidéo avec les sous-titres incrustés et un son externe

```
gst-launch filesrc location=video.ogv ! oggdemux name=demux \
filesrc location=music.oga \
! queue ! oggdemux ! vorbisdec ! audioconvert ! vorbisenc ! mux. demux. \
! queue ! theoradec !  subtitleoverlay name=sub !  theoraenc ! \
mux. filesrc location =./video.srt !  subparse ! sub. oggmux name=mux ! \
filesink location=videoOutput.ogv
```

Listing 7 – Enregistrer une vidéo avec l'audio `music.oga` l'image `video.ogv` et les sous-titres

3 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, mais j'ai choisi de rester proche de l'ogg, étant donné la nature des vidéos que vous nous avez fournis depuis le début de ce module.

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 ne m'en suis pas servis.

3.1 Question 5

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

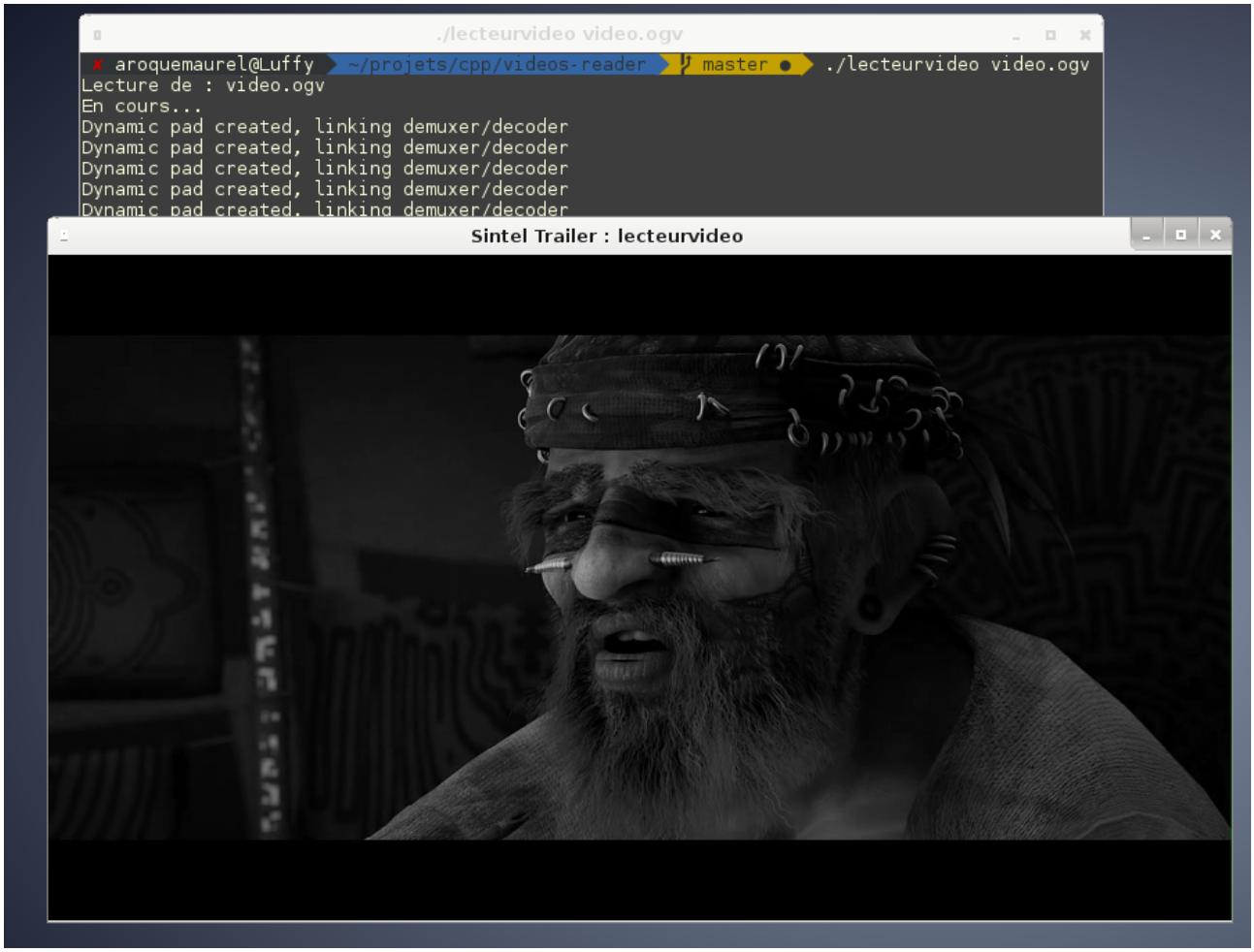

Figure 1 – Lecteur vidéo sans interface

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

J'ai cependant modifié légèrement le code fournit, dorénavant, j'ai créé une classe permettant de fournir des méthodes pour tous les `GstElement`. L'interface de cette classe me permet d'éviter d'avoir des dizaines de variables déclarés [1], de vérifier que tout a été alloué correctement et de tout ajouter automatiquement au pipeline, ceux en un minimum de lignes.

```
class GStreamerCommands {
public:
  GStreamerCommands(void);
  ~GStreamerCommands(void);

  void setElement(std::string nameElement, const std::string nameArg, const std::string ↩
      valuePropertie);
  void setElement(std::string nameElement, const std::string nameArg, const double ↩
      valuePropertie);
  void setElement(std::string nameElement, const std::string nameArg, const bool ↩
      valuePropertie);
  GstElement *getElement(std::string s);
  void addElement(std::string name, std::string value);

  void checkAllElements(void);
  void addAllElements(void);
  GstElement *getPipeline(void) const;
  void setPipeline(GstElement *getPipeline);
};
```

Listing 8 – Interface de gestion des commandes GStreamer

1. Ceci grâce à une `std::map`

3.2 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 :

```
// Add all srt elements
_commands->addElement ("subOverlay",  "subtitleoverlay");
_commands->addElement ("subSource",   "filesrc");
_commands->addElement ("subParse",    "subparse");

// Linkage
gst_element_link_many (_commands->getElement("videoQueue"), ↩
    _commands->getElement("videoDecoder"),
_commands->getElement("videoConv"), _commands->getElement("subOverlay"),
_commands->getElement("videosink"), NULL);
gst_element_link_many (_commands->getElement("audioQueue"), ↩
    _commands->getElement("audioDecoder"),
_commands->getElement("audioConv"), _commands->getElement("audiosink"), NULL);
g_signal_connect (_commands->getElement("demuxer"), "pad-added", G_CALLBACK ↩
    (on_pad_added), _commands->getElement("audioQueue"));
g_signal_connect (_commands->getElement("demuxer"), "pad-added", G_CALLBACK ↩
    (on_pad_added), _commands->getElement("videoQueue"));
```

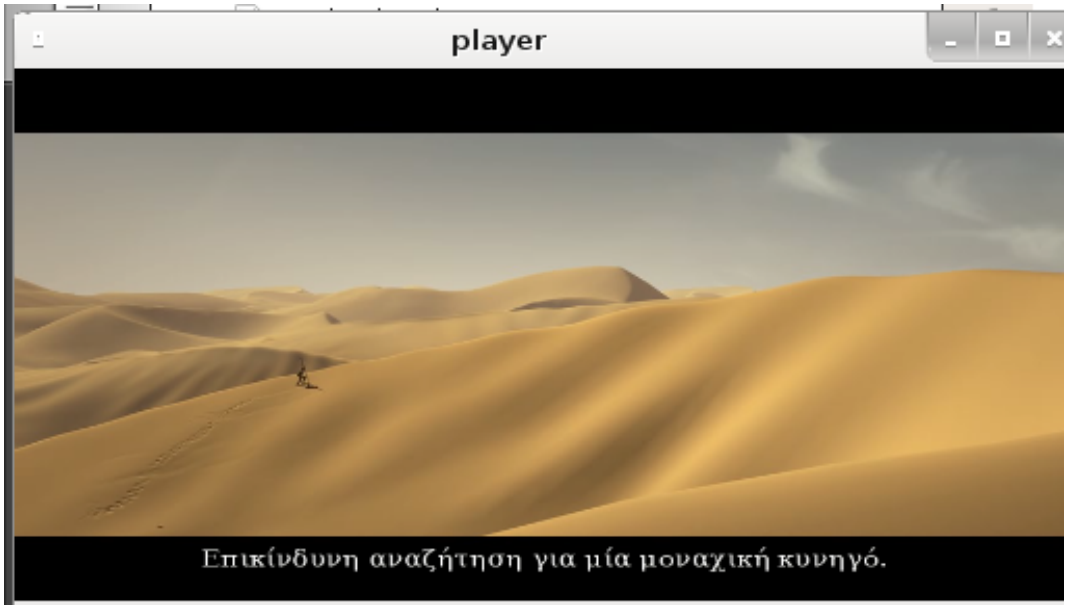

Figure 2 – Lecteur vidéo avec sous-titres sans interface

3.3 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.

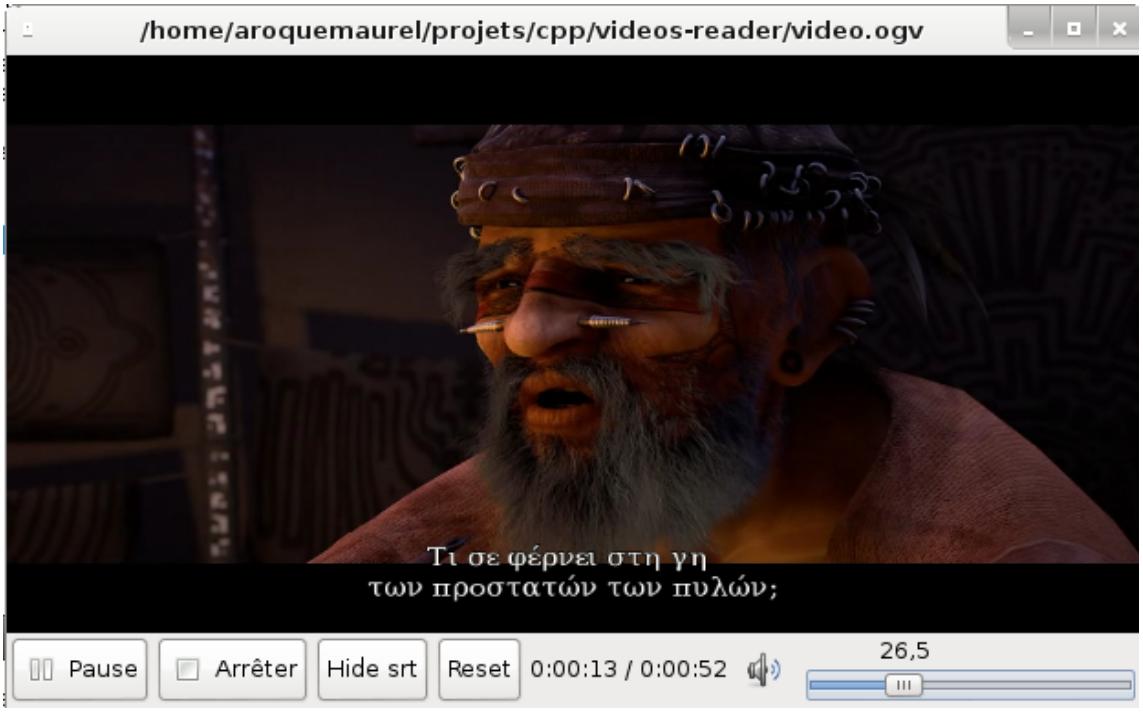

Figure 3 – Interface du lecteur multimédia développé

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);
```

3.4 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 et de contrôler la vidéo

– Un label affichant la durée de la vidéo, et le temps actuel parcouru. [2]
– Une possibilité de mettre la vidéo en plein écran, à l'aide de la touche « F » ou « F11 ».
– Gestion du son via le bouton de son. Celui-ci va de 0 à 2 fois le son original de la vidéo.
J'ai également mis en place des raccourcies à l'aide du clavier afin que l'utilisateur gagne au maximum du temps :

Touche du clavier	Action
F ou F11	Met la vidéo en Met la vidéo en plein écran
Barre d'espace ou P	Pause ou play en fonction de l'état courant de la vidéo
S	Affiche ou cache les sous-titres en fonction de l'état actuel des sous-titres
R	Redémarre la vidéo
Q	Quitte le lecteur vidéo
« Backspace » / Retour arrière	Stop la lecture de la vidéo
Flèches gauche et droite	Recule ou avance respectivement dans la vidéo
Flèche haut et bas	Augmente ou diminue le volume sonore

3.5 Activer ou désactiver les sous-titres

L'utilisateur peut activer ou non les sous-titres à l'aide du bouton situé en bas de l'interface.

Afin de pouvoir afficher ou non les sous-titres, il m'a suffit d'utiliser l'argument `silent` de `subtitleoverlay`.

```
// Désactive les sous-titres
g_object_set (G_OBJECT (_commands->getElement("subOverlay")), "silent", false, NULL);
// Active les sous-titres
g_object_set (G_OBJECT (_commands->getElement("subOverlay")), "silent", true, NULL);

// Ou avec mon interface
_commands->setElement("subOverlay", "silent", false);
```

Listing 9 – Affichage ou non des sous-titres

3.6 Reconnaissance des sous-titres

Ce principe est actuellement utilisé dans les vidéos *youtube* par Google, au vu des nombreuses erreurs qu'effectue ce système, on peut penser que cela est particulièrement compliqué à mettre en place.

En effet, beaucoup de facteurs rentrent en ligne de compte, notamment la vitesse d'élocution, l'accent, la grammaire etc...

Néanmoins, nous pourrions penser à un système qui possède une reconnaissance vocale, analyse une phrase et l'injecte automatiquement dans le flux de données afin d'en créer le sous-titre. La difficulté étant donc ce système de reconnaissance vocale.

A Listings

2. Au format hh :mm :ss, à l'aide de la macro `GST_TIME_FORMAT`

B Table des figures