

Programando Aplicações Multimídia no GStreamer

Guilherme Lima **Rodrigo Costa** Roberto Gerson

Pontifícia Universidade Católica – PUC-Rio
Lab. TeleMídia, Dep. de Informática.

{gflima,rodrigocosta,robertogerson}@telemidia.puc-rio.br

8 de novembro de 2016



Quem somos

Guilherme Lima

- Doutor em Informática pela PUC-Rio (2015)
- Interesses de Pesquisa: linguagens de programação e modelos para sincronismo multimídia.

Rodrigo Costa

- Doutorando em Informática na PUC-Rio
- Interesses de Pesquisa: sistemas multimídia distribuídos e modelos síncronos para autoria multimídia.

Roberto Gerson

- Doutor em Informática pela PUC-Rio (2015)
- Interesses de Pesquisa: representação e autoria de cenas multimídia interativas e representação e renderização de vídeos 3D.

Acessando o Repositório do Minicurso

<https://github.com/TeleMidia/minicurso-webmedia16>

- Pasta *src*
 - códigos fonte utilizados neste minicurso
- Pasta *slides*
 - slides utilizados neste minicurso

Sumário

1. Introdução
2. Hands on
 - Olá Mundo
 - Player MP3
 - Player Ogg
 - Comandos gst-launch e gst-inspect
 - Filtros
 - Play, pause, stop, seek, fast-forward, rewind
3. Plugins
4. Considerações Finais

Sumário

1. Introdução
2. Hands on
 - Olá Mundo
 - Player MP3
 - Player Ogg
 - Comandos gst-launch e gst-inspect
 - Filtros
 - Play, pause, stop, seek, fast-forward, rewind
3. Plugins
4. Considerações Finais

GStreamer

- Um dos principais *frameworks* de código aberto para processamento de dados multimídia
 - Projeto com mais de 15 anos
- Projetado facilitar o desenvolvimento de aplicações que apresentam ou processam conteúdo audiovisual
- Softwares que usam o GStreamer:
 - amaroK
 - Banshee
 - Eina
 - Empathy
 - Rhythmbox
 - Totem

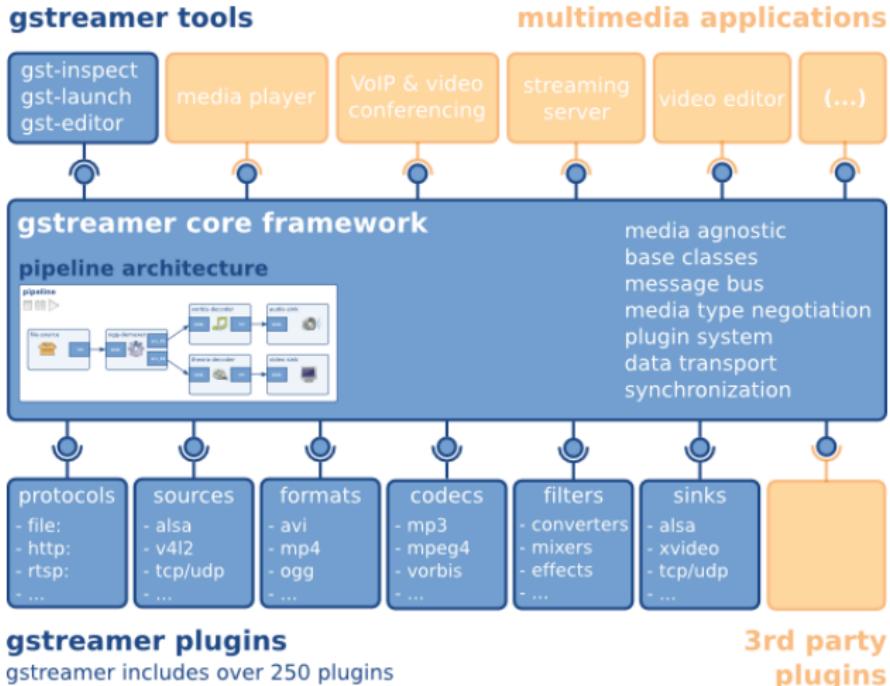
GStreamer

- Multiplataforma
- Robusto
- Flexível
- Extensível por Plugins
- APIs de alto e baixo nível
- Baseado no modelo de computação *dataflow*
 - “*Pipeline*” na terminologia do GStreamer
- Desenvolvido em C
 - Possui *bindings* para outras linguagens

O que o GStreamer NÃO é

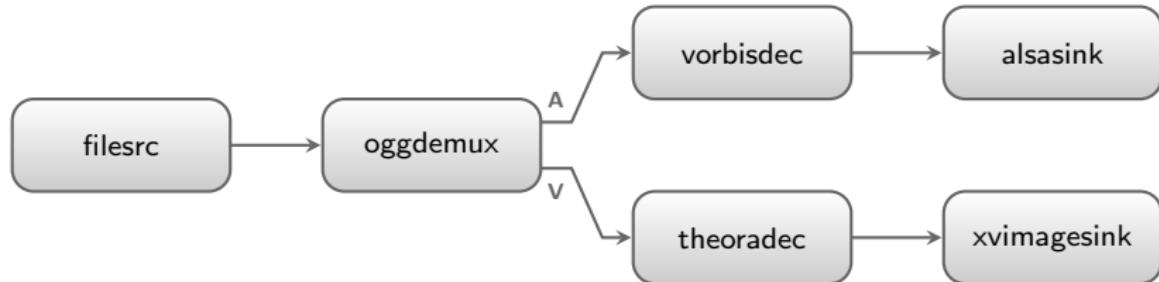
- Implementação de codec
- Aplicação *standalone*

GStreamer



Dataflow

- Dados são processados enquanto “fluem” através de uma rede
- Estrutura de grafo dirigido
 - Nós representam elementos de processamento (atores)
 - Arestas representam conexões unidirecionais por onde fluem os dados
- Atores recebem dados em portas de entrada e emitem dados através de portas de saída
- Um *pipeline* é um *dataflow* em que os dados fluem na mesma ordem em que foram produzidos

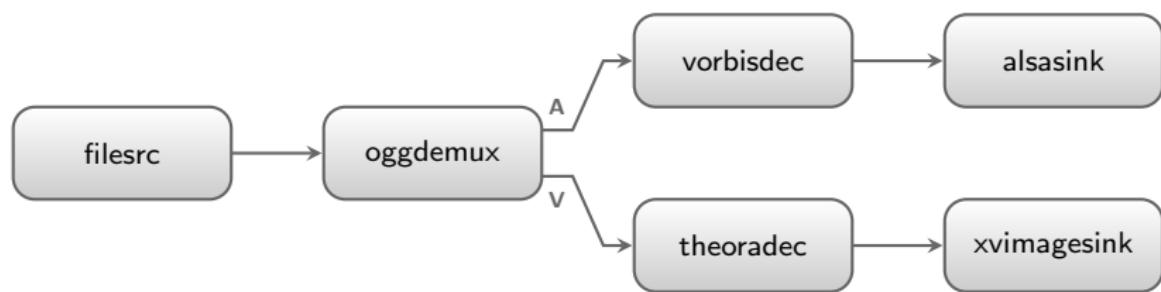


Pipelines no GStreamer

- Em um *pipeline* GStreamer os dados que fluem são tipicamente amostras de áudio e vídeo e dados de controle
- Nós do grafo (atores) são chamados elementos
- Portas por onde entram e saem dados dos elementos são chamados de *pads*
 - *Sink pad* – portas de entrada
 - *Source pad* – portas de saída
- Tipos de elementos
 - *Source* (produtores) – possuem apenas *source pads*
 - Processadores – possuem *source* e *sink pads*
 - *Sink* (consumidores) – possuem apenas *sink pads*

Pipelines no GStreamer

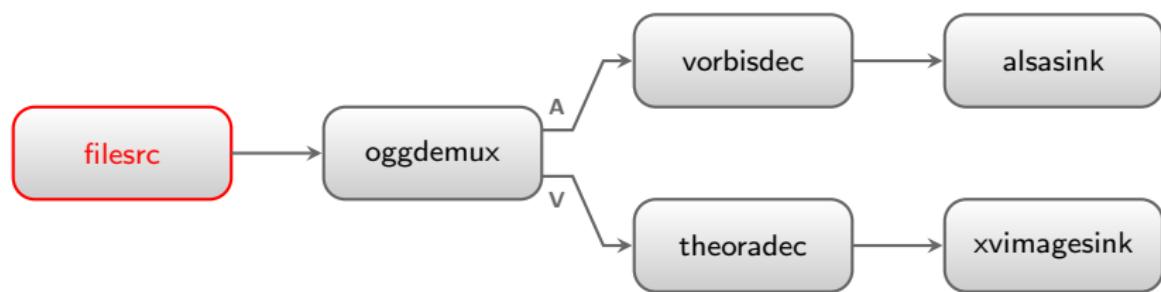
Um *pipeline* GStreamer simplificado que reproduz um arquivo de vídeo Ogg



Pipelines no GStreamer

Um *pipeline* GStreamer simplificado que reproduz um arquivo de vídeo Ogg

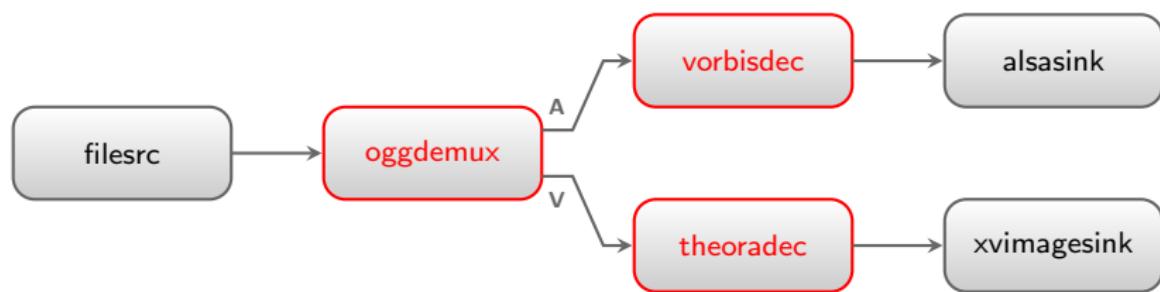
- *Source*
 - filesrc



Pipelines no GStreamer

Um *pipeline* GStreamer simplificado que reproduz um arquivo de vídeo Ogg

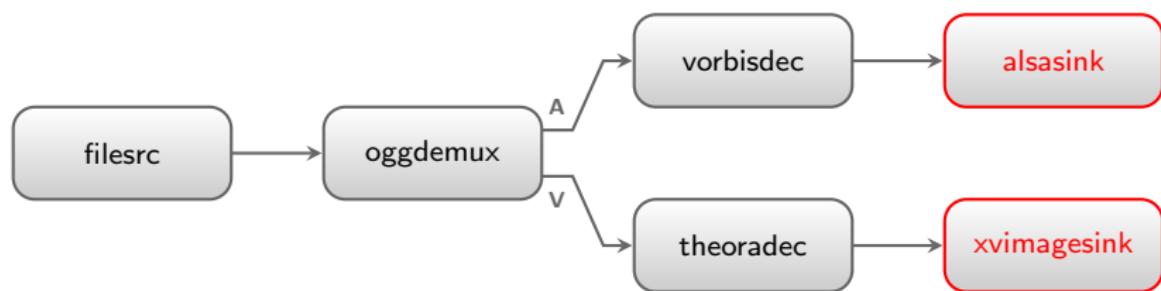
- Processadores
 - oggdemux
 - vorbisdec
 - theoradec



Pipelines no GStreamer

Um *pipeline* GStreamer simplificado que reproduz um arquivo de vídeo Ogg

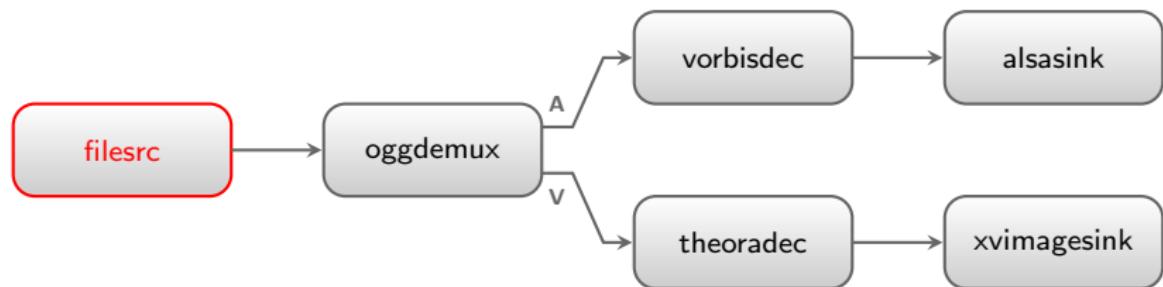
- *Sinks*
 - alsasink
 - xvimagesink



Pipelines no GStreamer

filesrc

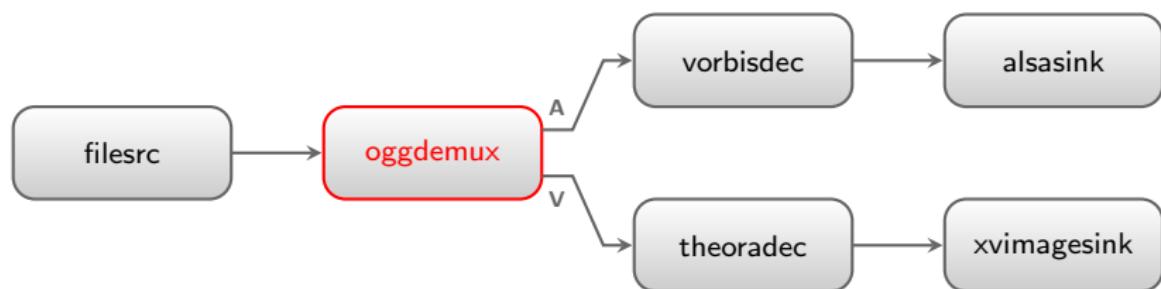
Lê um arquivo do disco (vamos assumir um arquivo Ogg) e escreve o fluxo de bytes resultante na sua *source pad*



Pipelines no GStreamer

oggdemux

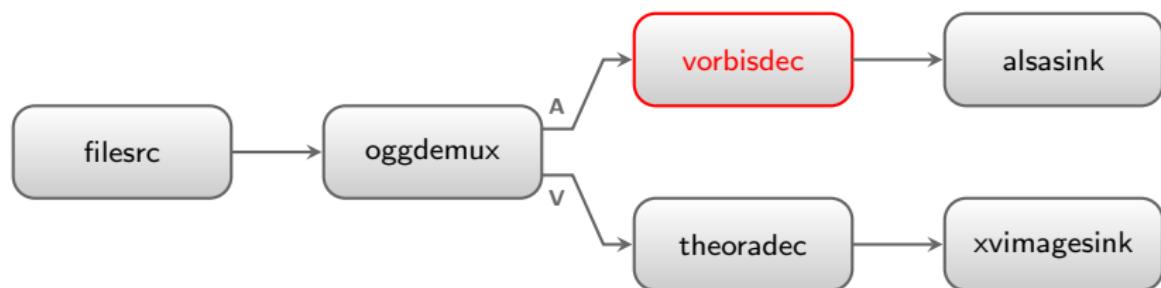
Lê da sua *sink pad* um fluxo de bytes codificado no formato Ogg, demultiplexa-o e escreve os fluxos Vorbis (áudio) e Theora (vídeo) resultantes nas *source pads* correspondentes



Pipelines no GStreamer

vorbisdec

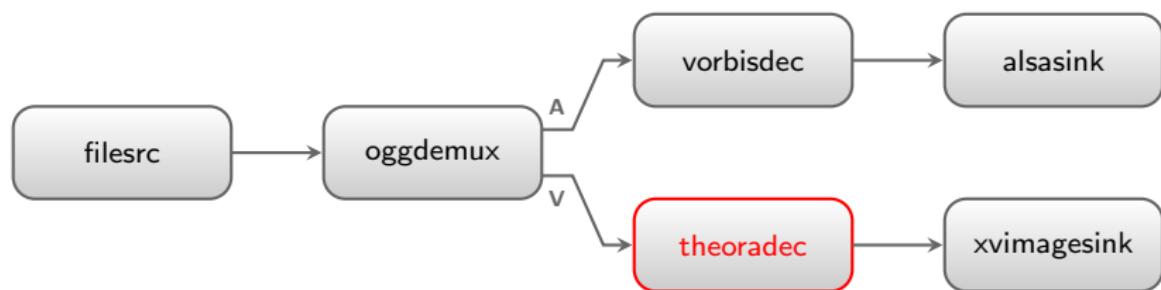
Lê da sua *sink pad* um fluxo de bytes codificado no formato Vorbis, decodifica-o e escreve o fluxo de áudio PCM resultante (áudio *raw* descomprimido) na sua *source pad*



Pipelines no GStreamer

theoradec

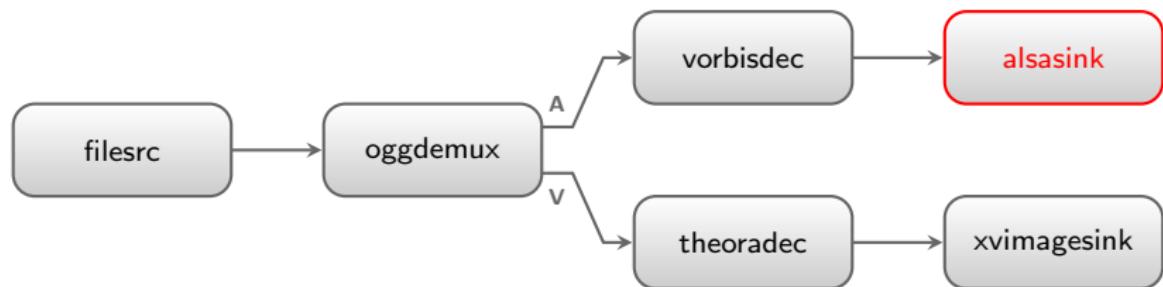
Lê da sua *sink pad* um fluxo de bytes codificado no formato Theora, decodifica-o e escreve o fluxo de vídeo *raw* descomprimido resultante na sua *source pad*



Pipelines no GStreamer

alsasink

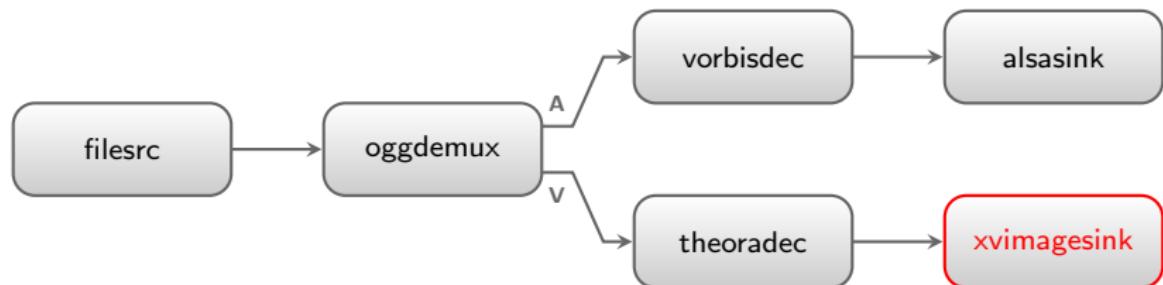
Lê um fluxo de áudio descomprimido da sua *sink pad* e utiliza a biblioteca ALSA para reproduzir as amostras do fluxo nos alto-falantes



Pipelines no GStreamer

xvimagesink

Lê um fluxo de vídeo descomprimido da sua *sink pad* e utiliza a biblioteca X11 para reproduzir os quadros do fluxo na tela



Sincronização

- *Pipelines* possuem um relógio para controlar a sincronização dos fluxos
- Cada amostra de áudio e vídeo possuem um tempo de apresentação (PTS – *presentation timestamp*) e uma duração
- Elementos *sink* controlam a taxa de reprodução de cada fluxo
 - Amostras recebidas antes do seu tempo de apresentação são armazenadas em uma fila interna para serem exibidas no tempo adequado
 - Amostras recebidas após o seu tempo de apresentação são descartadas
- Todos os outros elementos do *pipeline* operam livremente
 - Consumem e produzem dados em taxas arbitrárias

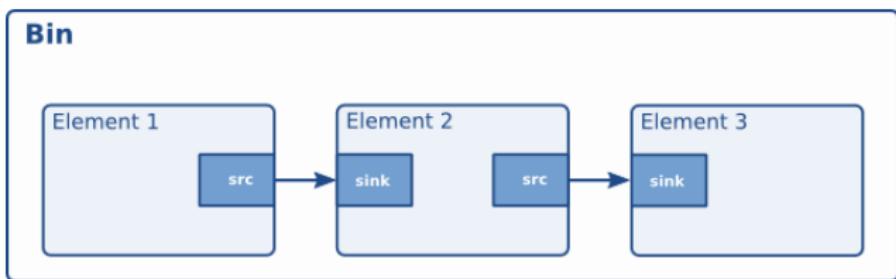
Pads

- Pads são pontos de conexão entre elementos
 - source → sink ✓
 - sink → source ✗
- Dois tipos de dados trafegam entre pads
 - Dados (*buffers*)
 - Amostras de áudio/vídeo
 - Fluem exclusivamente na direção das conexões
 - Eventos (*events*)
 - Informações de controle
 - Podem fluir em ambos os sentidos das conexões
 - Ex: QoS, seek, flush, ...
- Dados e eventos podem percorrer as conexões em paralelo



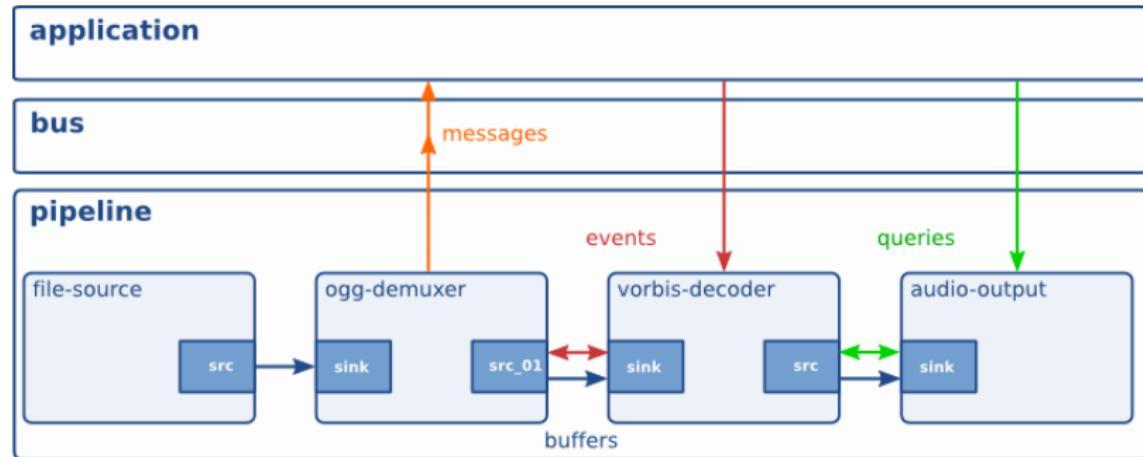
Estrutura conceitual de uma conexão entre pads no GStreamer.

Bins – Agrupando Elementos em Contêineres



- Bins são contêineres que agrupam elementos
 - *Pipelines* também são *bins*
- Dois elementos só podem ser ligados se estiverem dentro do mesmo contêiner

Bus – Barramento de Comunicação



- Elementos postam mensagens no barramento de comunicação (*bus*) para se comunicarem com a aplicação

Sumário

1. Introdução
2. Hands on
 - Olá Mundo
 - Player MP3
 - Player Ogg
 - Comandos gst-launch e gst-inspect
 - Filtros
 - Play, pause, stop, seek, fast-forward, rewind
3. Plugins
4. Considerações Finais

Sumário

1. Introdução

2. Hands on

Olá Mundo

Player MP3

Player Ogg

Comandos gst-launch e gst-inspect

Filtros

Play, pause, stop, seek, fast-forward, rewind

3. Plugins

4. Considerações Finais

Primeira Aplicação: Olá mundo

Tocando um vídeo no GStreamer usando o elemento “playbin”

- Arquivo: src/hello.c

Compilando o código fonte hello.c:

```
$ cc hello.c -o hello `pkg-config --cflags --libs  
glib-2.0 gstreamer-1.0`
```

Sumário

1. Introdução

2. Hands on

Olá Mundo

Player MP3

Player Ogg

Comandos gst-launch e gst-inspect

Filtros

Play, pause, stop, seek, fast-forward, rewind

3. Plugins

4. Considerações Finais

Tocando um Arquivo MP3

Player MP3

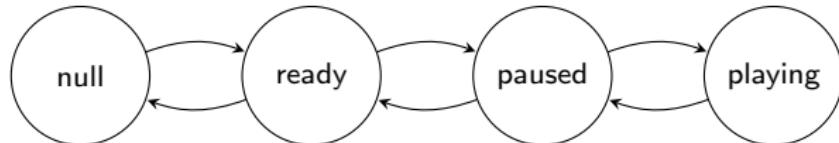


Pipeline que reproduz um arquivo de áudio MP3

- Arquivo: src/mp3.c

Máquinas de Estado de um GstElement

- Todo elemento possui um estado, que pode ser:
 - GST_STATE_NULL (nulo)
 - GST_STATE_READY (pronto)
 - GST_STATE_PAUSED (pausado)
 - GST_STATE_PLAYING (tocando)
- Mudanças de estado são propagadas para elementos filhos (*bins*)
 - Direção de propagação é sempre dos *sinks* para os *sources*
 - Isso evita que dados sejam gerados antes que elementos posteriores no *pipeline* estejam prontos para recebê-los



Sumário

1. Introdução

2. Hands on

Olá Mundo

Player MP3

Player Ogg

Comandos gst-launch e gst-inspect

Filtros

Play, pause, stop, seek, fast-forward, rewind

3. Plugins

4. Considerações Finais

Um pouco mais sobre *pads*

Cada *pad* possui três atributos importantes de serem considerados para ligar dois elementos

Um pouco mais sobre *pads*

Cada *pad* possui três atributos importantes de serem considerados para ligar dois elementos

- Direção
 - *Sink*
 - *Source*

Um pouco mais sobre *pads*

Cada *pad* possui três atributos importantes de serem considerados para ligar dois elementos

- Direção
 - *Sink*
 - *Source*
- Capacidade
 - *Caps* – determina os tipos de *buffers* que podem atravessá-la

Um pouco mais sobre *pads*

Cada *pad* possui três atributos importantes de serem considerados para ligar dois elementos

- Direção
 - *Sink*
 - *Source*
- Capacidade
 - *Caps* – determina os tipos de *buffers* que podem atravessá-la
- Disponibilidade
 - *Always* (sempre)
 - *Sometimes* (às vezes)
 - *Request* (sob requisição)

Disponibilidade de *Pads*

- *Always*
 - Criadas assim que o elemento é criado
 - `gst_element_get_static_pad ()`

Disponibilidade de *Pads*

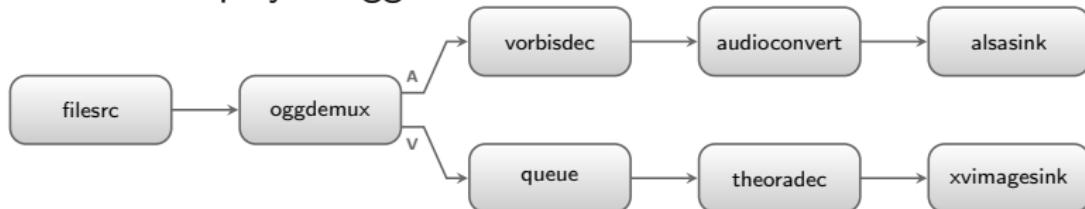
- *Always*
 - Criadas assim que o elemento é criado
 - `gst_element_get_static_pad ()`
- *Sometimes*
 - Criada pelo próprio elemento sob condições específicas que normalmente envolvem o conteúdo processado

Disponibilidade de *Pads*

- *Always*
 - Criadas assim que o elemento é criado
 - `gst_element_get_static_pad()`
- *Sometimes*
 - Criada pelo próprio elemento sob condições específicas que normalmente envolvem o conteúdo processado
- *Request*
 - Criada apenas quando requisitadas pela aplicação
 - `gst_element_get_request_pad()`

Tocando um Arquivo Ogg

Pipeline de um player Ogg



- Um contêiner Ogg pode conter múltiplos fluxos multiplexados
- O demultiplexador `oggdemux` cria *source pads* para cada fluxo multiplexado no arquivo
 - *Source pads* possuem disponibilidade *sometimes*
- Só podemos conectar o elemento `oggdemux` ao resto do *pipeline* quando as *source pads* forem criadas
- Notificação assíncrona (sinais)
 - O elemento `oggdemux` emite o sinal *pad-added* sempre que uma nova *source pad* é criada

Tocando um Arquivo Ogg

Player Ogg:

- Arquivo: src/ogg2.c

Sumário

1. Introdução
2. Hands on
 - Olá Mundo
 - Player MP3
 - Player Ogg
 - Comandos gst-launch e gst-inspect
 - Filtros
 - Play, pause, stop, seek, fast-forward, rewind
3. Plugins
4. Considerações Finais

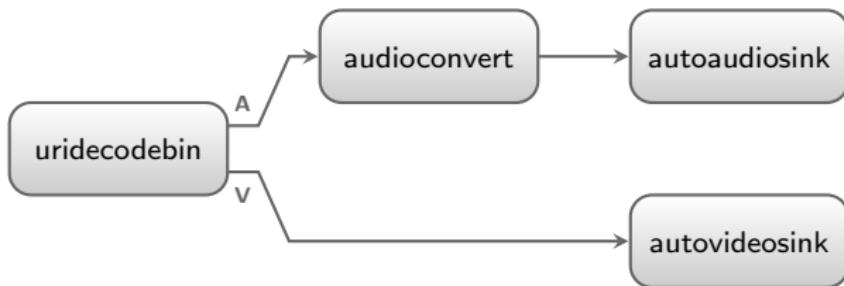
gst-launch

- Permite construir pipelines estáticos (que não mudam após montados) tipo diretamente na linha de comando
- Particularmente útil para depurar ou testar a viabilidade de *pipelines* antes de implementá-los em C
- Em algumas instalações o nome do comando é `gst-launch-1.0`

```
gst-launch playbin uri="file://$PWD/bunny.ogg"
gst-launch filesrc location=bunny.mp3 ! mad !
    alsasink
gst-launch filesrc location=bunny.ogg ! oggdemux
    name=demux\
        demux. ! vorbisdec ! audioconvert ! alsasink\
        demux. ! queue      ! theoraadec      ! xvimagesink
```

Um *Pipeline* Genérico

```
gst-launch uridecodebin uri="<URI>" name=decbin \
decbin. ! audioconvert ! autoaudiosink \
decbin. ! autovideosink
```



gst-inspect

- Permite inspecionar os *plugins* e elementos disponíveis no sistema
- Em algumas instalações o nome do comando é *gst-inspect-1.0*

Exemplo: \$ `gst-inspect equalizer`

```
1 Plugin Details:
2   Name                  equalizer
3   Description           GStreamer audio equalizers
4   Filename              /usr/lib/gstreamer-1.0/libgstequalizer.so
5   Version               1.8.3
6   License               LGPL
7   Source module          gst-plugins-good
8   Source release date   2016-08-19
9   Binary package         GStreamer Good Plugins (Arch Linux)
10  Origin URL            http://www.archlinux.org/
11
12  equalizer-nbands: N Band Equalizer
13  equalizer-3bands: 3 Band Equalizer
14  equalizer-10bands: 10 Band Equalizer
15
16  3 features:
17    +- 3 elements
```

Sumário

1. Introdução

2. Hands on

Olá Mundo

Player MP3

Player Ogg

Comandos gst-launch e gst-inspect

Filtros

Play, pause, stop, seek, fast-forward, rewind

3. Plugins

4. Considerações Finais

Filtros

- Aplicam transformações sobre um fluxo de amostras descomprimidas
- Alteram o fluxo original, produzindo efeitos sonoros ou visuais

Alguns filtros de áudio e vídeo disponíveis no GStreamer:

	Plugin	Elemento	Descrição (controle)
Áudio	audiofx	audiodynamic	Compressão ou expansão
	audiofx	audioecho	Eco
	equalizer	equalizer-nbands	Equalização (<i>n</i> bandas)
	freverb	freverb	Reverberação
	soundtouch	pitch	Tonalidade e tempo
	speed	speed	Velocidade
	volume	volume	Volume
.....			
Vídeo	coloreffects	coloreffects	Efeito de colorização
	effectv	revtv	Efeito de relevo
	effectv	shagadelictv	Efeito de espiral caleidoscópica
	videocrop	videocrop	Recorte
	videofilter	videobalance	Brilho, cor e saturação
	videoscale	videoscale	Redimensionamento

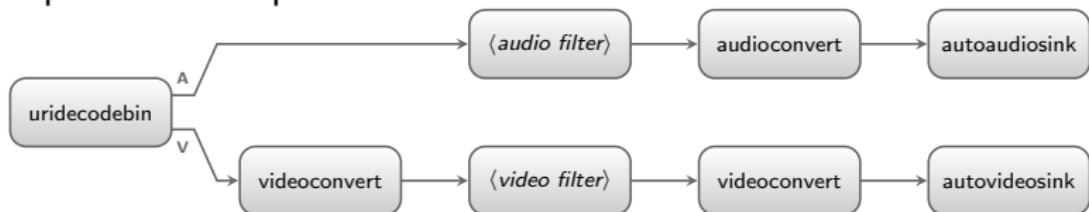
Aplicando filtros

Exemplo simples usando gst-launch:

```
$ gst-launch filesrc location=bunny.mp3 \
    ! mad ! volume volume=.5 ! alsasink
```

Aplicando filtros

Exemplo mais complexo:



- Arquivo: src/filter.c

Aplicando filtros

Efeitos de vídeo aplicados pelo programa anterior:



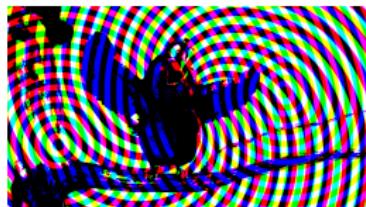
(a) Quadro original.



(b) Filtro “coloreffects”.



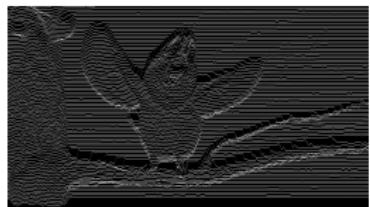
(c) Filtro “dicetv”.



(d) Filtro “shagadelictv” .



(e) Filtro “edgetv” .



(f) Filtro “revtv” .

Sumário

1. Introdução

2. Hands on

Olá Mundo

Player MP3

Player Ogg

Comandos gst-launch e gst-inspect

Filtros

Play, pause, stop, seek, fast-forward, rewind

3. Plugins

4. Considerações Finais

Controles avançados

Vamos incrementar o programa “Olá mundo” para que ele responda aos seguintes comandos:

-  SPC pausa ou resume a reprodução (*pause e play*)
-  → avança 5s (*seek*)
-  ← retrocede 5s (*seek*)
-  F reproduz 2x mais rápido (*fast-forward*)
-  R reproduz ao contrário 2x mais rápido (*rewind*)
-  N reproduz na velocidade original
-  Q para a reprodução e termina o programa (*stop*)

Controles avançados

- Elementos *sink* postam mensagens de eventos de navegação `GST_NAVIGATION_MESSAGE_EVENT` no *bus* para informar eventos de tecla e de mouse
- A postagem de eventos *seek* (`GST_EVENT_SEEK`) requisita mudanças na posição e na taxa de reprodução de um fluxo
- Arquivo: `src/controls.c`

Sumário

1. Introdução
2. Hands on
 - Olá Mundo
 - Player MP3
 - Player Ogg
 - Comandos gst-launch e gst-inspect
 - Filtros
 - Play, pause, stop, seek, fast-forward, rewind
3. Plugins
4. Considerações Finais

Plugins no GStreamer

- Plugin: componente de software que estende funcionalidades de um programa/*framework*
- Elementos GStreamer são encapsulados em *plugins*
 - Um *plugin* pode implementar diversos elementos
- *Plugins* geram bibliotecas dinâmicas (.dll, .so) que são carregadas na chamada `gst_init()`
- A API de *plugins* é fortemente baseada no sistema de tipos do framework GLib/GObject

GLib/GObject

GLib

- *Framework* multiplataforma da GNOME
- Implementação em C de diversas estruturas de dados (listas encadeadas, tabelas hash, árvores binárias, ...)
- Sistema de tipos dinâmico (GType)
- Implementação de orientação a objetos em C (GObject)

GLib/GObject

GLib

- Framework multiplataforma da GNOME
- Implementação em C de diversas estruturas de dados (listas encadeadas, tabelas hash, árvores binárias, ...)
- Sistema de tipos dinâmico (GType)
- Implementação de orientação a objetos em C (GObject)

GObject

- Tipo fundamental (superclasse) da GLib
- Gerenciamento de memória (contador de referências)
- Funções para construção/destruição de instâncias
- Sistema genérico de propriedades
- Sinais

Elemento “myfilter”

- Filtro genérico que lê uma amostra da sua *sink pad*, imprime o seu tamanho na saída padrão e repassa-a à sua *source pad*
- Arquivo: src/myfilter2.c

Compilação:

```
$ cc myfilter.c -o myfilter.so -shared -fPIC
`pkg-config --cflags --libs gstreamer-1.0
gstreamer-base-1.0`
```

Teste:

```
$ gst-launch --gst-plugin-path=.
videotestsrc ! myfilter ! ximagesink
```

Elemento “myvideofilter”

Efeito de ondulação produzido pelo filtro “myvideofilter”.



(g) $factor = 0.01$



(h) $factor = 0.05$



(i) $factor = 0.1$

Elemento “myvideofilter”

- Filtro de vídeo
 - As *pads* desse filtro só aceitam amostras de vídeo
- Herda do tipo `GstVideoFilter`
 - Superclasse que trata as tarefas comuns envolvidas no processamento de quadros de vídeo
 - O GStreamer possui tipos especializados para construção de *sources* (`GstBaseSrc`), *sinks* (`GstBaseSink`), filtros em geral (`GstBaseTransform`), filtros de áudio (`GstAudioFilter`), etc.
- Aplica um efeito de ondulação aos amostras que o atravessam:

$$x(u, v) = u + 20 \times \sin(\text{factor} \times v)$$

$$y(u, v) = v,$$

Ellemento “myvideofilter”

- Arquivo: src/myvideofilter2.c

Compilação:

```
$ cc myvideofilter.c -o myvideofilter.so -shared -fPIC `pkg-config --cflags --libs gstreamer-1.0 gstreamer-base-1.0 gstreamer-video-1.0
```

Teste:

```
$gst-launch --gst-plugin-path=. \
    uridecodebin uri=<URI> \
    ! videoconvert \
    ! myvideofilter factor=.10 \
    ! videoconvert \
    ! autovideosink
```

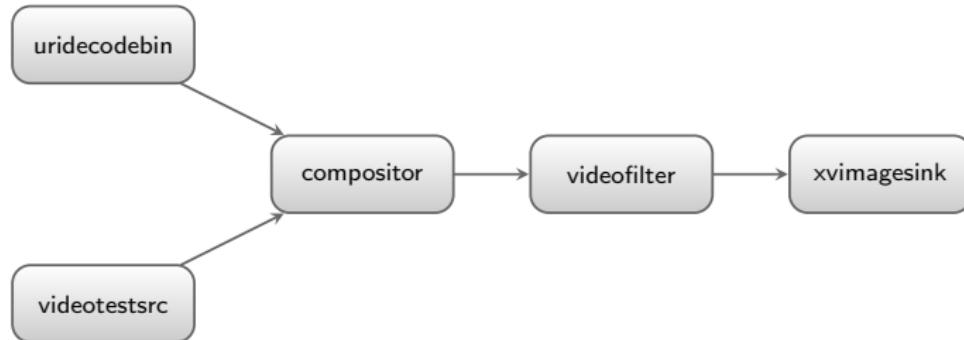
Sumário

1. Introdução
2. Hands on
 - Olá Mundo
 - Player MP3
 - Player Ogg
 - Comandos gst-launch e gst-inspect
 - Filtros
 - Play, pause, stop, seek, fast-forward, rewind
3. Plugins
4. Considerações Finais

Considerações Finais

Compondo múltiplos fluxos de vídeo num mesmo pipeline

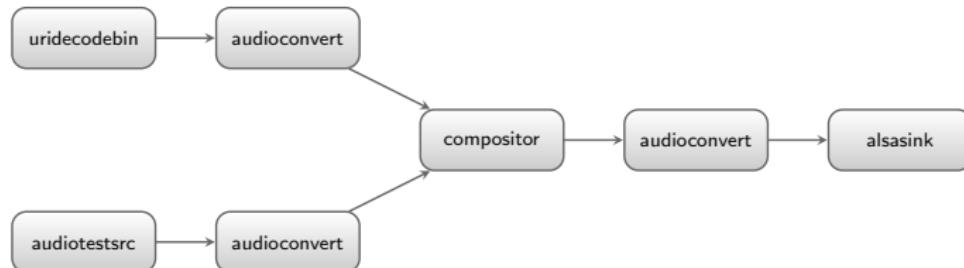
```
$ gst-launch uridecodebin uri=<URI> ! \
    compositor name=comp \
        sink_0::zorder=2 \
        sink_0::xpos=100 \
        sink_0::ypos=100 \
        sink_1::zorder=1 !
    videoconvert ! xvimagesink \
videotestsink ! comp.
```



Considerações Finais

Compondo múltiplos fluxos de áudio num mesmo pipeline

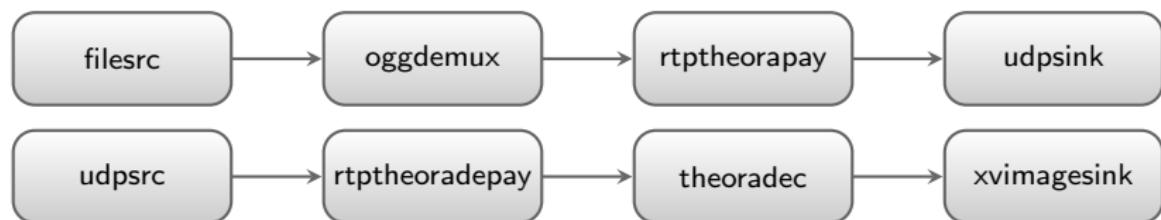
```
$ gst-launch uridecodebin uri=<URI> !\n    audioconvert ! audio/x-raw,rate=48000 !\\n\n    audiomixer name=mix !\n    audioconvert ! alsasink\n    audiotestsrc !\n    audioconvert ! audio/x-raw,rate=48000 !\\n\n    mix .
```



Considerações Finais

Transmissão de vídeos Ogg/Theora usando o protocolo RTP:

Servidor



Cliente

Considerações Finais

- Sincronização de relógios entre *pipelines*
 - `GstNetTimeProvider` + `GstNetClientClock`
- Sincronização de relógios com um servidor NTP (*Network Time Protocol*)
 - `GstNtpClock`
- Sincronização de relógios com um servidor PTP (*Precision Time Protocol*)
 - `GstPtpClock`
- *Bindings* para outras linguagens/plataformas
 - Lua, Python, Java, C++, Qt, Android, Vala, Ryby, Haskell,

Obrigado!