Universidade de São Paulo

Instituto de Matemática e Estatística

MAC499 - Trabalho de Formatura Supervisionado

Visualizador de Múltiplos Vídeos

Alunos:
Edson Kenji Ninomiya
José Antonio C. Marum Jr.

Orientador:
Prof. Roberto Hirata Jr.

1 de dezembro de 2009

Sumário

Agradecimentos

Ao Thiago T. Santos pelas orientações iniciais para o andamento do projeto.

Aos professores do IME-USP, cujos ensinamentos dados durante a graduação foram de grande importância para a realização deste trabalho.

E aos amigos e familiares, pelo apoio e incentivos constantes.

Capítulo 1

Introdução

No mundo de visão computacional, frequentemente aplicamos algoritmos de processamento de imagens para que se possa analisá-las e extrair informações para aplicações posteriores. Diversos algoritmos podem ser aplicados sobre uma mesma entrada, obtendo assim diferentes modificações da mesma.

Eventualmente, gostaríamos de comparar as saídas obtidas com as imagens originais, ou mesmo entre elas mesmas, a fim de descobrir se os processamentos aplicados cumprem com o seu objetivo. Nesse caso, usualmente, colocam-se as imagens desejadas uma ao lado da outra e a comparação é feita visualmente pelo usuário.

Às vezes, queremos aplicar esses algoritmos em um vídeo, ou melhor nos quadros de um vídeo. Nesta situação o cenário se mostra muito mais complexo pois, para comparar os resultados dos processamentos realizados, é necessário desmontar o vídeo, cada um de seus *frames* sendo armazenado como uma imagem no disco rígido. Entretanto, há dois empecilhos neste procedimento: o primeiro é o espaço requerido para guardar todos os quadros, o segundo é a falta de praticidade no processo de comparação.

Um arquivo de vídeo, geralmente, é codificado de forma a otimizar o espaço exigido em disco sem perder muita qualidade nas imagens. Ao desmontar um vídeo em *frames*, perdemos essa propriedade.

Apesar de a inspeção poder ser automatizada, isso exigiria um novo processamento cuja função seria apenas verificar se o anterior cumpre com o seu objetivo. Em algum momento, a comparação deverá ser realizada visualmente pelo usuário, colocando as imagens lado a lado em um trabalho tedioso de reposicionamento e redimensionamento de janelas.

Este trabalho propõe uma ferramenta que resolva estes problemas, através de uma interface amigável, permitindo ao usuário executar *frame* a *frame* diversos vídeos de uma só vez, facilitando assim esse processo de inspeção.

A seguir, serão apresentados os conceitos básicos necessários para os próximos capítulos. Depois, veremos as ferramentas que foram utilizadas durante o desenvolvimento do *Visualizador de Múltiplos Vídeos*. Mais tarde, discorreremos a respeito de suas funcionalidades. Finalmente, um capítulo detalhando os resultados obtidos, seguido pela conclusão e futuro deste trabalho.

I. Parte Técnica

Capítulo 2

Conceitos

Antes de discorrermos sobre a ferramenta implementada, primeiramente, a compreensão de alguns conceitos faz-se necessária. Ao longo desta sessão, veremos alguns destes fundamentos.

2.1 Imagem

A formação de uma imagem ocorre quando a interação entre alguma forma de radiação com objetos físicos é registrada por um sensor. Para capturar uma imagem digital, processo conhecido como digitalização, precisamos primeiramente de um modelo matemático que defina certos padrões, de modo que possamos guardar essa imagem como um conjunto de dados [6]. Todo modelo matemático usado para esse fim possui as seguintes características:

- *Função de Imagem*: é a abstração fundamental de uma imagem;
- *Modelo Geométrico*: descreve como três dimensões são projetadas em duas;
- *Modelo Radiométrico*: mostra como o sensor é afetado pelas propriedades do objeto e da radiação (como reflexão, geometria, etc.);
- *Modelo de Frequência Espacial*: descreve como variações espaciais da imagem devem ser performadas;
- *Modelo de Coloração*: descreve como diferentes espectros de cores são relacionados com as cores originais da imagem;
- *Modelo de Digitalização*: descreve o processo de obtenção de áreas da imagem de forma discreta.

2.2 Frame

Historicamente, os primeiros vídeos eram gravados em uma longa fita de filme fotográfico, onde era possível ver cada uma das imagens que compunham a animação como uma foto emoldurada (em inglês, **frame**), de onde deriva o nome.

Um frame é uma imagem $f \in E \rightarrow K^3$, onde E é um subconjunto de $\mathbb{Z}$x$\mathbb{Z}$ e K é um intervalo de $\mathbb{N}$, com $K = [0, k]$, $k = 255$ ou 65535.

2.3 Vídeo

Um vídeo é uma sequência ordenada de frames f_i, $i \in [0, \ldots, \mathbb{N}]$.

Quando um vídeo é executado, cada frame é exibido por um curto intervalo de tempo. Um fenômeno conhecido como *Persistência Retiniana* é o responsável por criar a ilusão de movimento em nossos cérebros.

Atualmente, esse intervalo de tempo gira em torno de 240, 250 ou 300 milissegundos. Assim sendo, a cada segundo são exibidos aproximadamente de 24 a 30 frames (a esta cadência, damos o nome de *frame rate*).

2.4 Codec

Codec vem do acrônimo de "***Co****mpressor-****Dec****ompressor*", muito embora os primeiros codecs não tivessem realmente a função de compressão. Originalmente, sua função consistia em codificar e decodificar os arquivos de mídia, de forma que dados analógicos fossem transformados em dados digitais. Atualmente, os codecs podem ser divididos em dois grupos: *lossless* e *lossy* (do inglês, sem perdas e com perdas, respectivamente).

Os codecs do tipo *lossless* têm como função comprimir os arquivos sem provocar alterações. Isso significa que o arquivo ao ser descomprimido será idêntico ao original. Alguns exemplos deste tipo de codec incluem o *flac* e o *wavpack* para arquivos de áudio, o *PNG* e o *TIFF* para imagens, e o *FFmpeg* para vídeos.

Em contrapartida, codecs do tipo *lossy* sacrificam a qualidade do arquivo original em troca de uma compressão muito maior. Enquanto codecs *lossless* comprimem o arquivo de duas a três vezes, os do tipo *lossy* chegam a deixar o arquivo com cerca de um décimo, ou menos, do tamanho original. As alterações provocadas por eles geralmente não são perceptíveis sob condições normais. Alguns exemplos deste tipo de codec incluem o *MP3* e o *WMA* para arquivos de áudio, o *Jpeg* e o *GIF* para imagens, e o *Xvid* e o *WMV* para vídeos.

2.5 Processamento de Imagem

Processamento de imagem consiste em manipular os dados de uma imagem A, de modo a obtermos uma imagem A' como saída [10]. Diferentemente do que ocorre com o *tratamento de imagens*, que apenas as altera para fim de apresentação, o processamento de imagens tem como objetivo obter informações para uso posterior em outras aplicações.

Dentre as diversas operações que podem ser performadas, podemos citar:

- *Segmentação de Imagem*: utilizada para extrair elementos de uma cena, como um rosto ou um texto. Preocupa-se em localizar regiões de fronteira, como as bordas de um objeto;
- *Diferença de Imagens*: bastante utilizada na astronomia para a detecção de corpos através do desvio da luz, consiste em sobrepor duas fotografias semelhantes e fazer uma "subtração" pixel a pixel;
- *Transformação Euclidiana*: trata de uma série de transformações na imagem, como redimensionamento, rotação e cisalhamento. Tem como objetivo manter as proporções da foto durante as operações aqui mencionadas.

Após o processamento, é possível utilizar o resultado para os mais diferentes fins, dentre os quais a detecção de movimento, a realidade aumentada e a análise biomédica (como a detecção de um tumor, ou a ampliação de uma imagem microscópica).

Capítulo 3

Bases Tecnológicas

Antes de iniciar a implementação do *Visualizador de Múltiplos Vídeos*, foi preciso decidir de antemão quais ferramentas seriam necessárias para o seu desenvolvimento.

A plataforma, bem como todas as outras bases tecnológicas utilizadas, foram escolhidas tendo como base o público alvo: a comunidade de visão computacional. Esse fator foi determinante em diversos pontos do projeto, como veremos em mais detalhes nos capítulos seguintes.

Ao longo deste capítulo, temos uma breve descrição das tecnologias utilizadas no projeto.

3.1 Python

Python [5] é uma linguagem de programação orientada a objetos interpretada que pode ser usada para a realização de uma grande variedade de tarefas, desde pequenos *scripts* até o desenvolvimento de aplicações inteiras. Sua simplicidade, quando comparada a outras linguagens, a torna uma das mais utilizadas por programadores iniciantes [7].

Figura 3.1: Python Logo

Além disso, Python é conhecido por ser uma ferramenta para o desenvolvimento rápido de aplicações: não é raro ouvirmos falar de projetos Python que são finalizados em dias ou mesmo horas, ao passo que essas mesmas aplicações levariam semanas ou meses em outras linguagens. Apesar de não ser tão rápida quanto as linguagens compiladas, como C e C++, quando pesamos o tempo despendido na criação do código, chegamos à conclusão de que na maioria das vezes utilizar Python é mais vantajoso [8].

Por essas razões, optamos por utilizá-lo no desenvolvimento da nossa aplicação.

3.1.1 Licença

A maior parte do código fonte do Python está protegida sob os termos da *Python Software Foundation* (PSF). Sua licença impõe poucas restrições de uso. De acordo com seus termos, a PSF garante ao licenciado o direito universal, não exclusivo e gratuito de reproduzir, testar, analisar, distribuir ou mesmo utilizar o Python ou qualquer versão derivativa; mas deixa claro que os termos da PSF ficam restritos ao próprio Python (ou à versão derivativa que estiver sendo usada).

3.2 GTK

GTK [3] é uma biblioteca multi-plataforma para criação de interfaces gráficas. O GTK foi inicialmente criado para o desenvolvimento do *GNU Image Manipulation Program* (GIMP), de onde deriva o seu nome (*GIMP Toolkit*); mas atualmente, ele é utilizado em uma ampla gama de projetos, entre eles, destaca-se o *GNU Network Object Model Environment* (GNOME).

Figura 3.2: GTK Logo

O GTK foi completamente desenvolvido em C, mas apesar disso, foi implementada utilizando os conceitos de classes e funções de *callback* (ponteiros para funções). Devido à sua popularidade, foram criados *wrappers* para diversas linguagens, dentre as quais Python, chamado de PyGTK.

Juntamente com o Qt, constituem os *toolkits* mais populares para o *X Window System*.

3.2.1 Por que GTK?

Dentre todas as bibliotecas para o desenvolvimento de interfaces gráficas em Python, quatro delas merecem destaque: Tk, Wx, Qt e GTK.

O Tk é a api gráfica padrão do Python. Ele foi descartado quase que imediatamente após o início do desenvolvimento do projeto, por se tratar de uma biblioteca muito simples, com poucos recursos. Já o Wx é uma api que atua diretamente na interface gráfica padrão do sistema operacional. Apesar de sua indiscutível portabilidade, ele apresenta algumas desvantagens, dentre elas a documentação pobre, instabilidade e problemas no desempenho.

O que nos fez optar pelo GTK ao invés do Qt foi o fato de o GTK ser baseado no modelo de objetos do *GLib 2.0*, também utilizado pelo GStreamer. Desta forma, o código seguiria um mesmo padrão, ficando assim mais coeso.

3.2.2 PyGTK

O PyGTK é um *wrapper* sobre a biblioteca de desenvolvimento de interfaces gráficas GTK para Python. As aplicações desenvolvidas com ele são verdadeiramente multi-plataforma, estando aptas a rodar sem qualquer modificação em qualquer sistema.

3.2.3 Licença

A biblioteca PyGTK segue os moldes da licença *GNU Lesser General Public License*, conhecida anteriormente como *GNU Library General Public License*.

3.3 GStreamer

GStreamer [2] é um arcabouço para desenvolvimento de aplicações de fluxo de mídia.

Figura 3.3: GStreamer Logo

Ele se utiliza de uma estrutura na qual diferentes *elementos* são ligados entre si, cada um responsável por processar os dados recebidos dos *elementos* anteriores e enviá-los para os seguintes, formando um grafo hierárquico.

Assim como o GTK, o GStreamer também foi desenvolvido inteiramente em C, possuindo um *wrapper* para Python, o PyGst. Ele também se baseia no modelo de objetos do *GLib 2.0*.

3.3.1 Estrutura

O **Elemento** é a classe mais importante no GStreamer. Existem diferentes tipos de elementos, cada um sendo responsável por apenas umas dentre as seguintes funções: executar a leitura dos dados de um arquivo, decodificar esses dados ou enviar os dados para uma saída (a placa de som ou de vídeo, por exemplo).

Os *elementos* podem ser ligados uns aos outros através de **Pads**. A saída de um *elemento* (*source pad*) conecta-se à entrada de outro (*sink pad*), de modo a permitir que os dados possam fluir através do grafo quando a *pipeline* estiver ativa (modo *playing*).

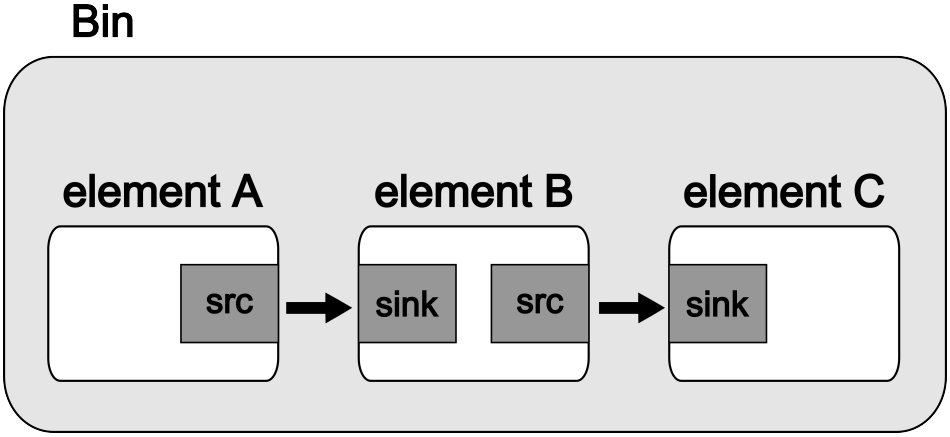


Figura 3.4: Estrutura - Figura Ilustrativa

Para combinar esses diferentes *elementos* entre si, precisamos de um contêiner, chamado de **Bin**. Uma **Pipeline** é um subtipo especial de *bin*, o qual permite a execução de todos os seus *elementos*. Uma vez que seja inicializada (colocada em estado *paused* ou *playing*), o fluxo de dados começa a ser executado.

3.3.2 Licença

O GStreamer é um software livre, licenciado sob os moldes da *GNU Lesser General Public License*.

Capítulo 4

Visualizador de Múltiplos Vídeos

Normalmente a comparação de imagens é feita entre duas delas dispostas lado a lado, como quando desejamos verificar o resultado de um processamento perante a foto original. No entanto, nada impede que mais de duas imagens sejam comparadas ao mesmo tempo. Às vezes, queremos analisar o resultado de vários processamentos diferentes aplicados sobre uma mesma imagem, a fim de identificarmos o mais apropriado para uma determinada situação. O espaço ocupado por todas essas fotos na tela precisa ser otimizado, de forma que cada imagem receba uma porção aproximadamente igual a das outras, para que a inspeção não seja comprometida.

Cada frame possui um quadro respectivo nos outros vídeos para comparação. Este processo é repetido diversas vezes e, por isso, é necessária uma certa organização para que não comprometamos os resultados dessas comparações, aplicando-a a frames errados.

Para tornar o cenário ainda mais complexo, os vídeos não precisam necessariamente serem iniciados no mesmo ponto. O processamento pode ser aplicado a partir de um determinado ponto de um vídeo, o que significa que para fazer a comparação, o intervalo entre esses vídeos deve ser levado em consideração.

A seguir, veremos como cada um desses problemas foi resolvido. Também veremos outros detalhes da aplicação.

4.1 Proposta da Nossa Aplicação

Como já havia sido mencionado no início do capítulo anterior, os usuários do *Visualizador de Múltiplos Vídeos* não seriam pessoas leigas no uso do computador, o que significa que muitos detalhes da interface foram planejados de forma a explorar isso.

4.1.1 Funcionalidades

Nesta sessão, serão mostrados *screenshots* da ferramenta sendo utilizada.

4.1.1.1 Primeira Tela

Esta é a tela inicial do programa em funcionamento. O design é bastante minimalista, para evitar que o usuário fique confuso durante sua utilização.

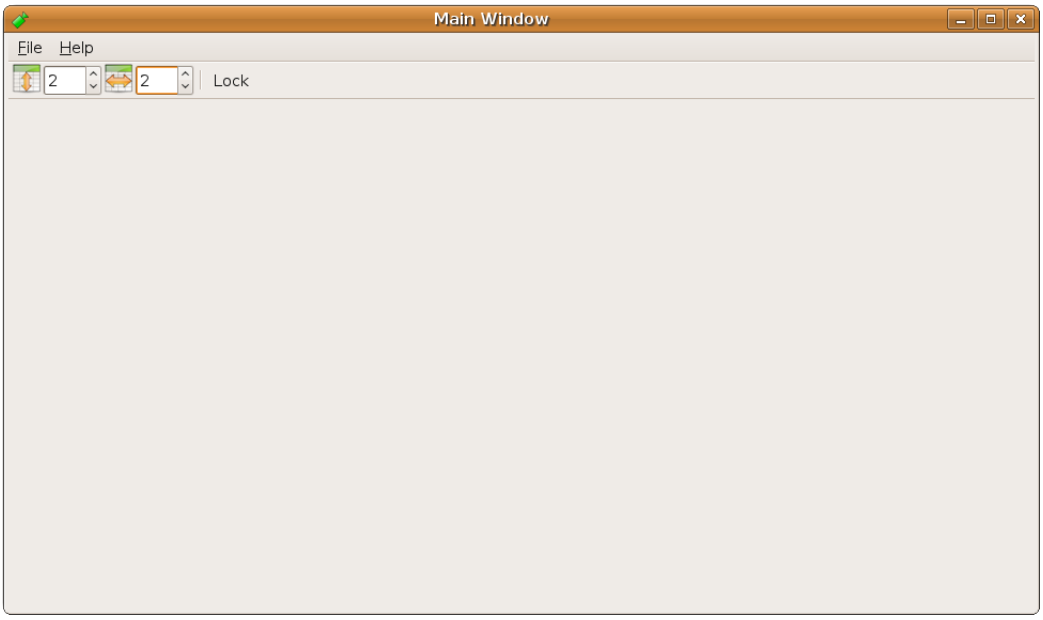
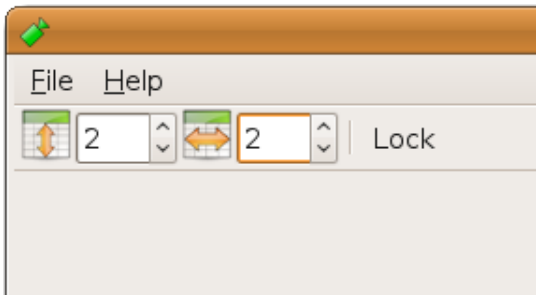


Figura 4.1: Primeira tela

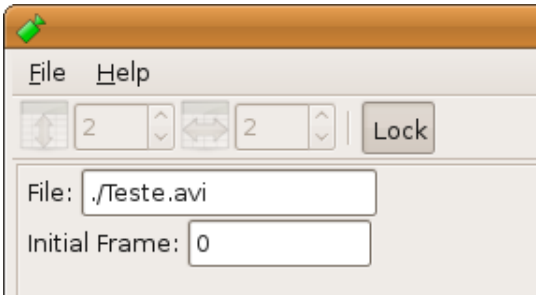
4.1.1.2 Seleção da Quantidade de Vídeos

O *Visualizador de Múltiplos Vídeos* oferece a opção de selecionar quantos vídeos se deseja comparar. Consiste de dois *spin buttons* na parte superior esquerda da tela inicial, cada um responsável pela quantidade de vídeos apresentada em um dos eixos da grade, sendo que o primeiro deles controla o número de linhas e o segundo o de colunas. Esse valor atualmente está limitado a cinco vídeos em cada eixo, o que totalizada 25 vídeos sendo mostrados ao mesmo tempo.

Após determinar o número de vídeos a serem apresentados, o usuário deve pressionar o *toggle button Lock*, localizado logo ao lado dos *spin buttons*. Quando ativado, ele desabilita a opção de seleção de quantidade de vídeos exibidos, e um painel para seleção de arquivos aparece logo abaixo. Ao desativá-lo, ocorre o inverso. Dessa forma, fica mais intuitivo para o usuário sobre como proceder com a ferramenta, além de simplificar bastante o cenário para tratamentos de erros.



(a) Quantidade de vídeos



(b) Seleção de quantidade de vídeos travada

As figuras acima mostram um exemplo desta funcionalidade sendo utilizada. No caso, o usuário está selecionando uma grade de apresentação para os vídeos de tamanho 2 por 2. E na imagem ao lado, a interface foi travada. Estes valores aqui escolhidos foram arbitrários. É importante notar que todos os *screenshots* mostrados daqui em diante terão exatamente a mesma configuração.

4.1.1.3 Seleção dos Arquivos de Entrada e Sincronização entre Vídeos

A ferramenta disponibiliza um painel em forma de grade, cada um dos quadrantes dela é preenchido por campos relacionados ao vídeo que se deseja exibir naquela posição da tela. Para tanto, o usuário deve informar o nome do arquivo e o seu frame inicial, como mostrado na figura abaixo:

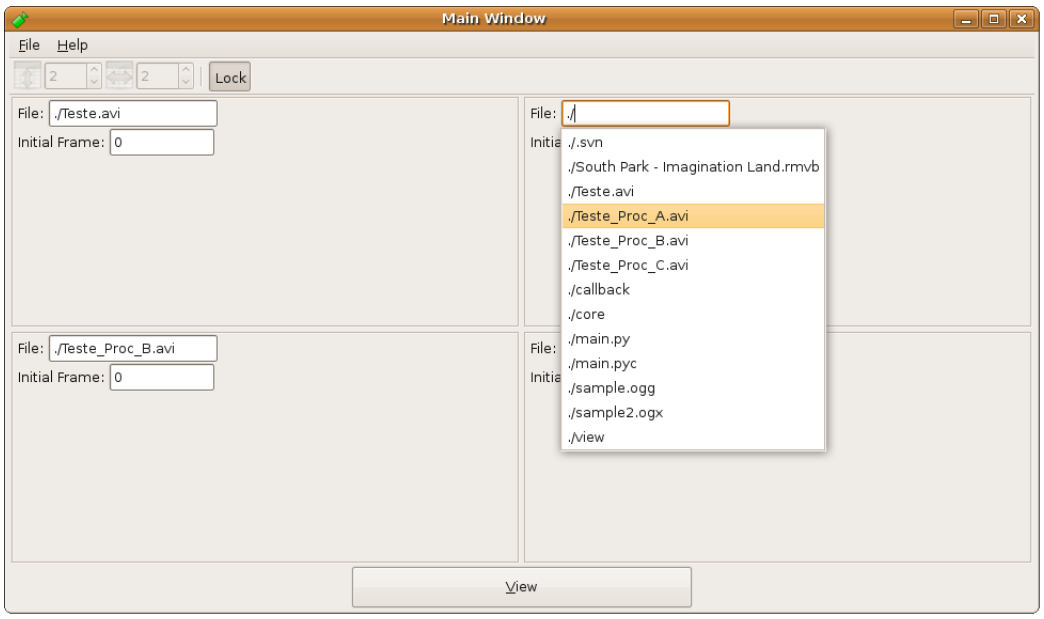


Figura 4.2: Seleção dos Arquivos de Entrada

Sendo o público alvo uma comunidade de uma determinada área da computação, optou-se por agilizar o processo de inserção dos dados de entrada. Ao invés de um botão no qual clicar e escolher o arquivo desejado, navegando pelas pastas, a ferramenta disponibiliza um campo de texto no qual o usuário pode inserir o caminho completo até o arquivo. Para facilitar esse processo, existe um sistema de *auto-completion*, que mostra todos os arquivos presentes no diretório atual, atualizado em tempo real, conforme o usuário digita o caminho, e completa o valor do campo automaticamente caso o usuário assim o queira.

Quanto ao campo do frame inicial, ele representa o valor da posição no qual o vídeo deve começar. O valor padrão é 0, que significa que o vídeo será iniciado a partir do primeiro frame. Este valor é utilizado pela tela de visualização dos vídeos (que será explicada na próxima sessão).

Depois de selecionados os vídeos e em qual quadro eles devem ser inicializados, clicamos no botão *View*, que se encontra na parte inferior da tela.

4.1.1.4 Apresentação dos Vídeos

Ao clicar no botão *View*, uma nova janela será aberta: a tela de visualização dos vídeos. Ela é a responsável pela apresentação dos vídeos ao usuário e, consequentemente, onde ele pode compará-los.

Determinou-se que os vídeos seriam apresentados ao usuário dispostos em uma grade. Isso resolveria a ambos os problemas citados no início deste capítulo: evitaria que o usuário se engane sobre qual *frame* pertence a qual vídeo, e também cada quadro seria mostrado na tela ocupando um espaço exatamente igual ao dos outros.

As figuras a seguir nos dão um vislumbre da tela de visualização:

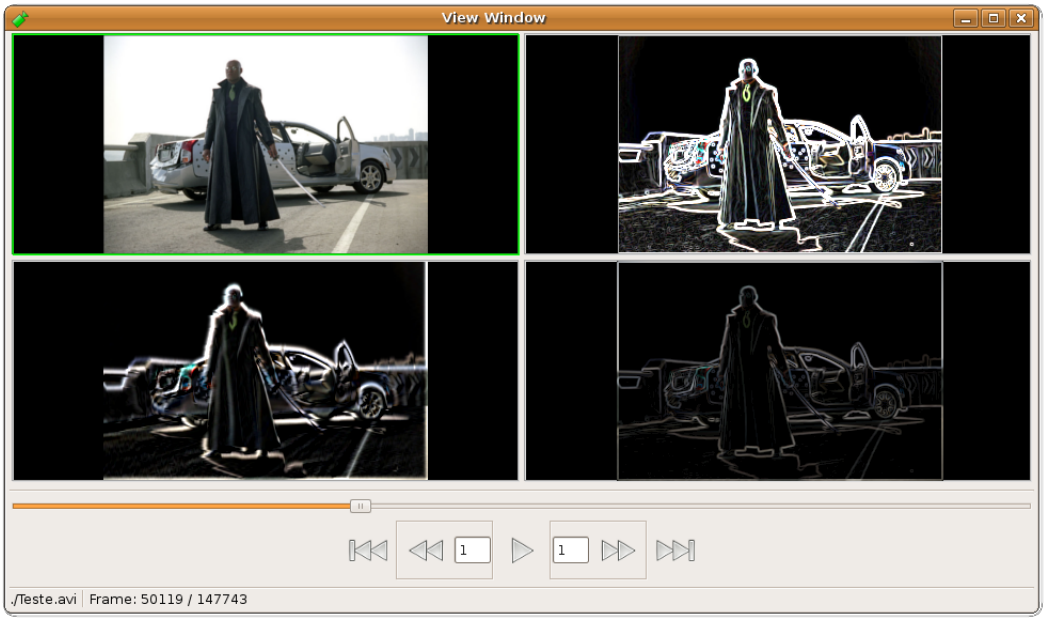


Figura 4.3: Seleção dos Arquivos de Entrada

Claramente notamos que a tela possui apenas um painel para a apresentação dos vídeos e um painel de controle de fluxo de mídia. Isso foi feito propositalmente, de modo a assegurar o máximo de espaço para os vídeos.

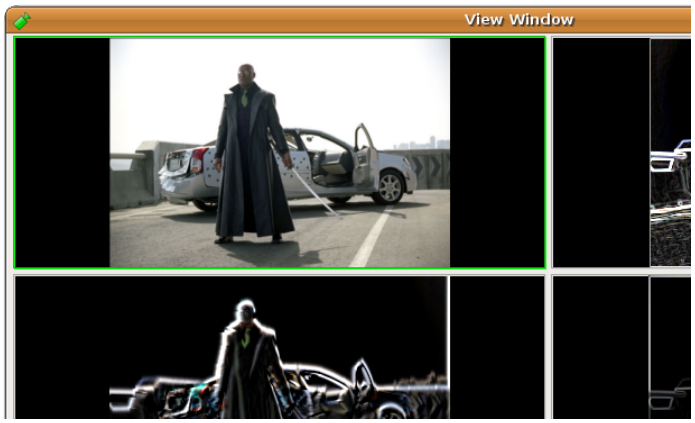


Figura 4.4: Vídeo em *Foco*

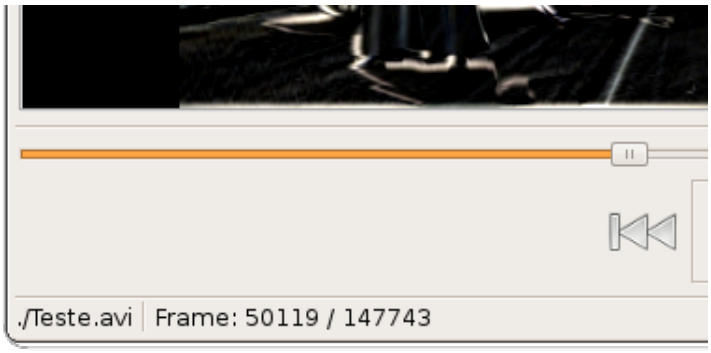


Figura 4.5: Informações do Vídeo

A figura 4.4 mostra o primeiro quadrante em destaque. É possível perceber um contorno verde ao redor dele, o que significa que ele está selecionado. Todas as informações apresentadas na parte inferior esquerda da tela, assim como a *trackbar*, estão atreladas ao vídeo do quadrante selecionado, como mostrado na figura 4.5.

A ferramenta determina que, por padrão, o vídeo ocupando o primeiro quadrante (o mais à esquerda e mais acima) é inicializado como sendo o selecionado. O usuário pode clicar em outro quadrante se desejar selecioná-lo e exibir as informações pertinentes a ele. As informações apresentadas incluem o nome do vídeo, em qual frame ele se encontra atualmente e quantos frames ele possui no total.

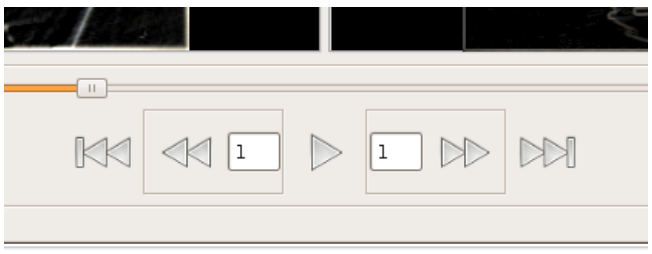


Figura 4.6: Controles do Visualizador

Por fim, no *screenshot* acima temos o painel de controle de fluxo de mídia, semelhante ao que pode ser encontrado na maioria dos tocadores convencionais. Entretanto, também existem duas caixas de texto, uma delas atrelada ao botão de avançar e a outra ao de retroceder.

Elas possibilitam ao usuário especificar quantos frames ele deseja que sejam passados ao clicar naquele botão. Por padrão, 1 frame é passado de cada vez.

Contudo, é importante ressaltar que qualquer botão pressionado afetará todos os vídeos. Isso significa que ao avançar X frames em um vídeo, X frames serão avançados em todos os demais. A mesma propriedade vale para o retrocesso. Uma validação no sistema impede que um vídeo vá para um quadro inválido (ou seja, para um frame anterior a 1 ou posterior ao número total de frames do vídeo).

Além disso, o botão que se encontra mais à esquerda no painel coloca todos os vídeos em suas respectivas posições iniciais, enquanto que o botão mais à direita os coloca em suas posições finais. Em ambos os casos, o *Visualizador de Múltiplos Vídeos* pausa o fluxo de dados.

A ferramenta ainda irá pausar o fluxo de dados sempre que algum vídeo chegue ao fim pois, de outra forma, teríamos uma perda na sincronização dos vídeos.

4.2 Implementação

Nesta sessão veremos como se deu o desenvolvimento da ferramenta.

4.2.1 Metodologias Empregadas

O desenvolvimento do projeto não seguiu a risca nenhuma das grandes metodologias conhecidas. Ao contrário, mesclamos algumas das práticas que julgamos serem as mais importantes, de modo a obter o que seria mais conveniente para a equipe.

De *Programação Extrema* adotamos a *programação pareada* (dois desenvolvedores em uma mesma máquina, um efetivamente programando enquanto é auxiliado pelo outro), o *simple design* (implementar exatamente as funcionalidades desejadas pelo cliente, se preocupando com a robustez do código implementado no futuro), o *ritmo sustentável* (trabalhando, em média, cerca de 20 horas por semana) e a *posse coletiva* (qualquer um da equipe pode alterar o código sem a permissão dos demais).

Já do *Scrum* adotamos a *daily scrum* (reuniões diárias realizadas logo no início do "expediente", que visam descobrir problemas durante a implementação de uma tarefa, repartir experiências, e focar o desenvolvedor para um determinada atividade para o restante daquele dia) e o *cross-functional team* (onde cada membro da equipe pode representar qualquer papel no desenvolvimento do software) [11].

4.2.2 Prototipagem

Durante o processo de implementação da ferramenta, foram feitos alguns protótipos possíveis de interface, chegando às imagens do protótipo de baixa fidelidade a seguir:

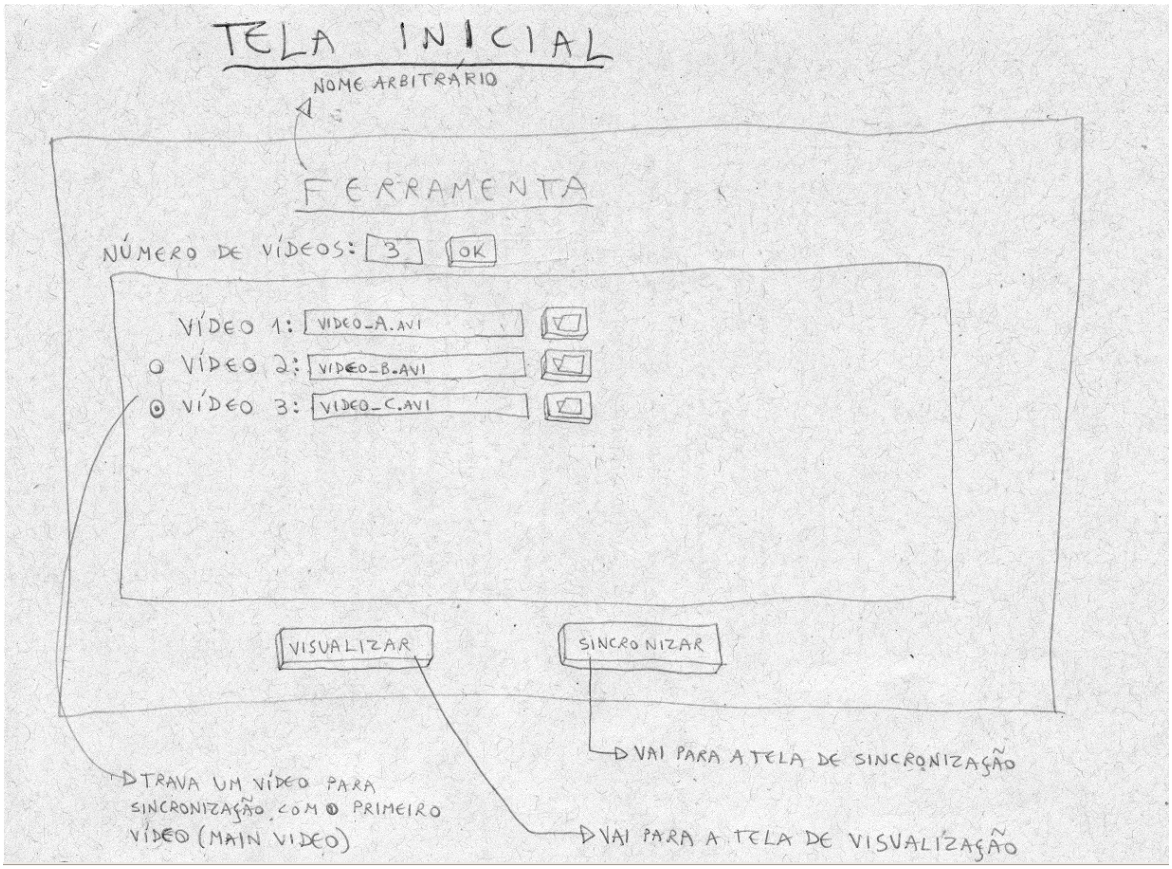


Figura 4.7: Protótipo de baixo nível - Tela Inicial

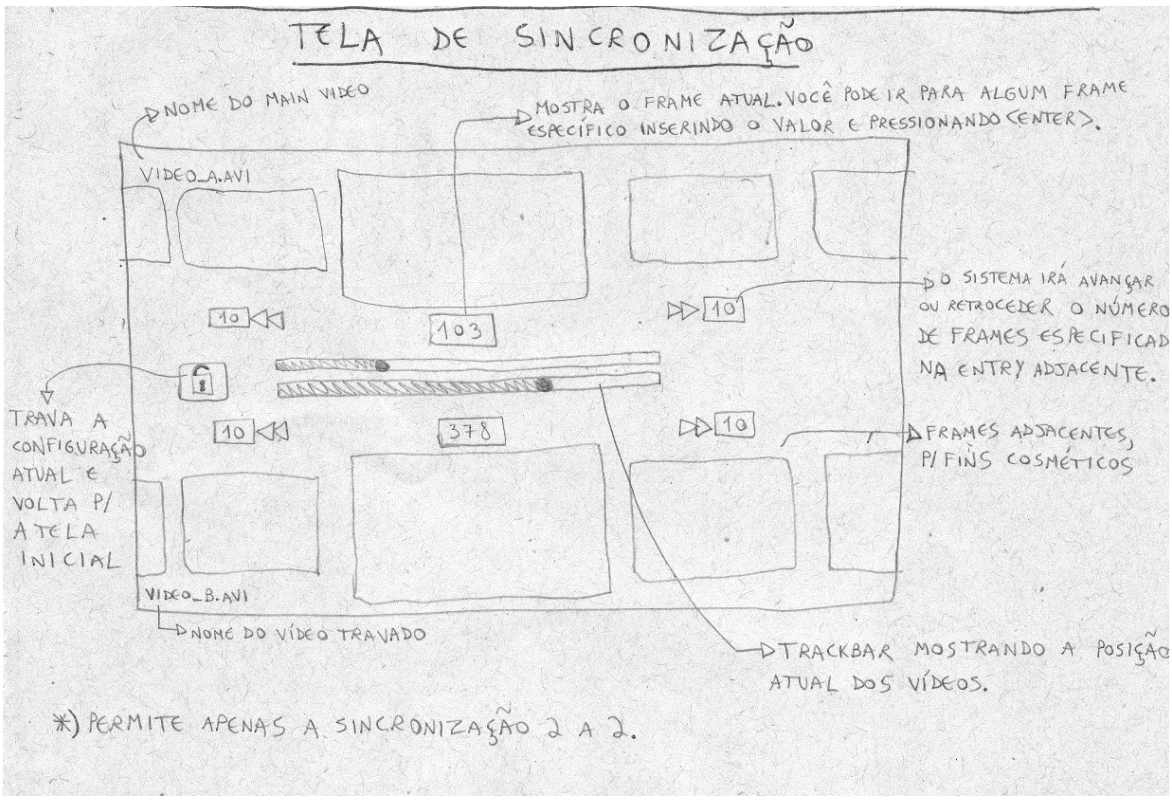


Figura 4.8: Protótipo de baixo nível - Tela de Sincronização

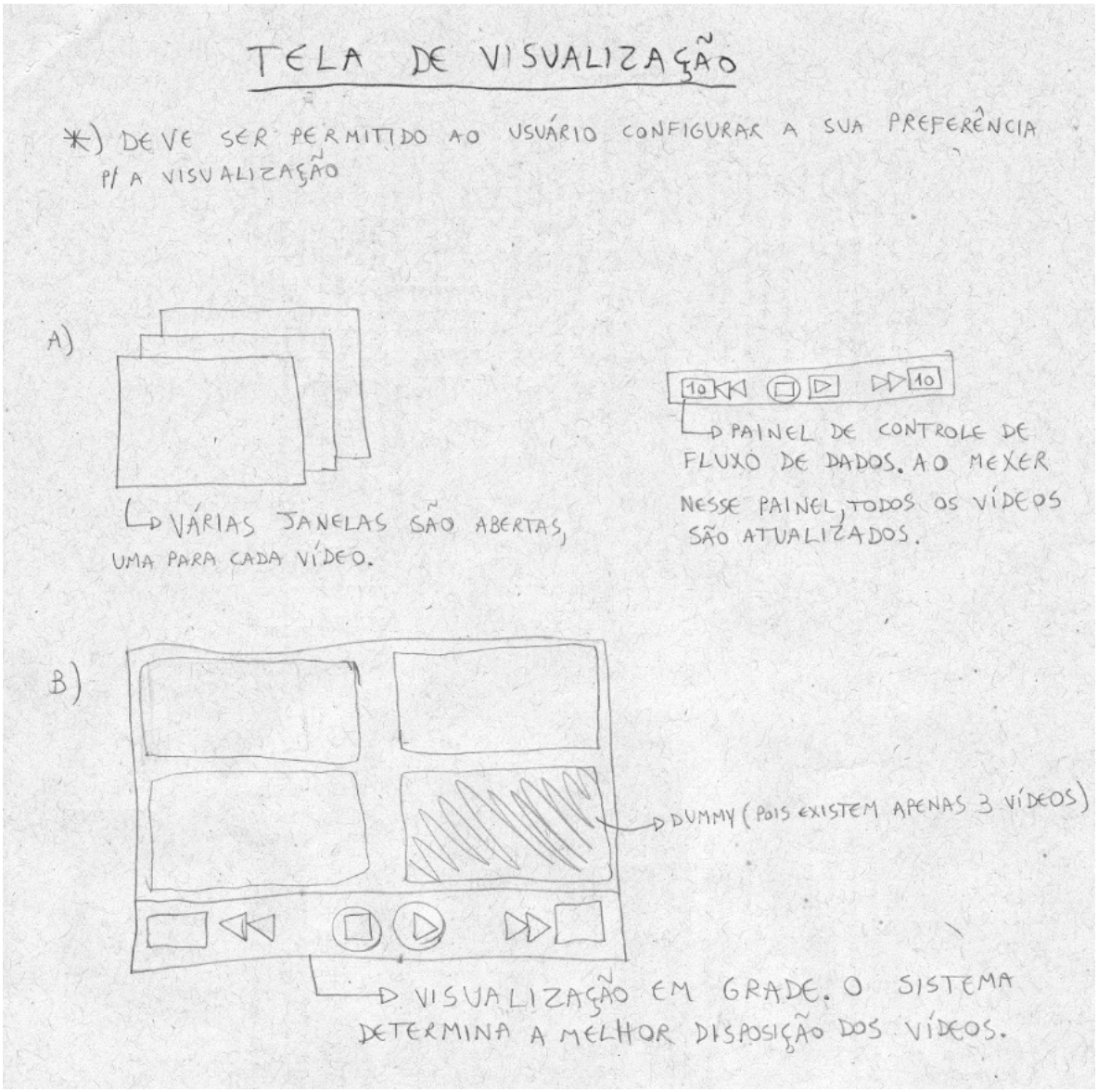


Figura 4.9: Protótipo de baixo nível - Tela de Visualização

Com o tempo, o protótipo de baixo nível foi sendo lapidado, conforme novas idéias iam surgindo e dificuldades técnicas aparecendo, como podemos ver no protótipo de média fidelidade abaixo:

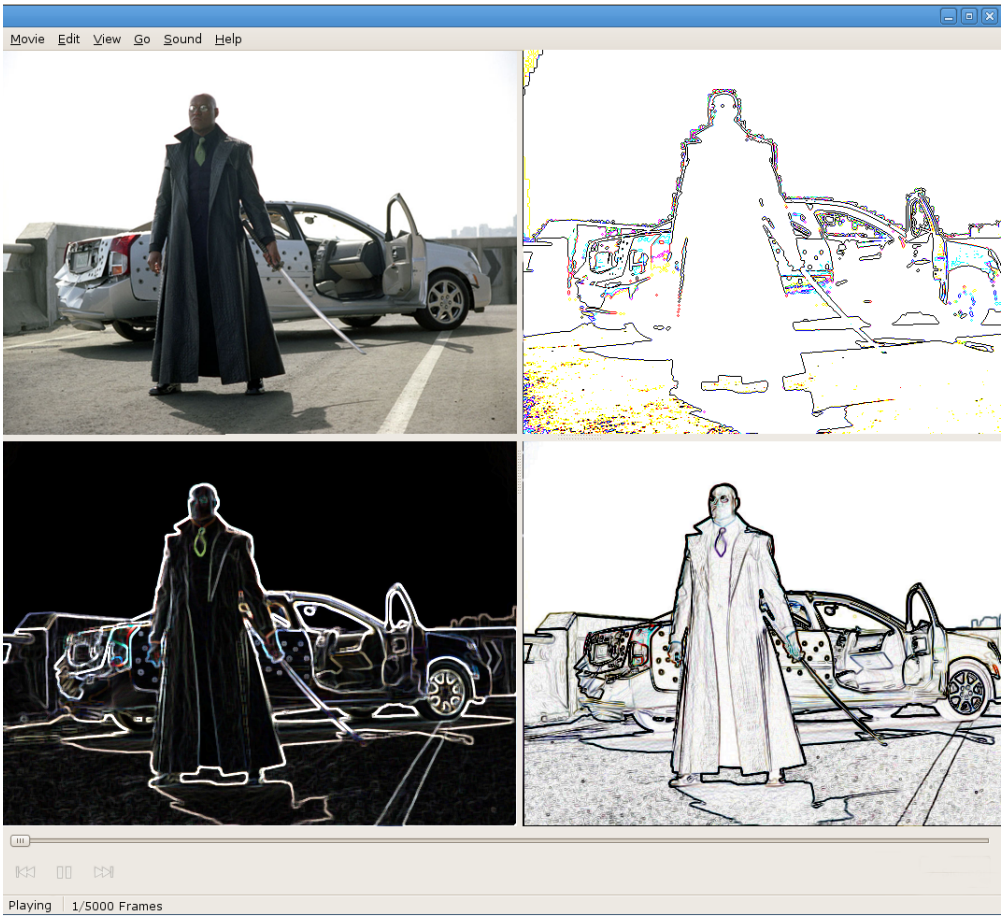


Figura 4.10: Protótipo de médio nível

A partir deste último, foram feitas mudanças em alguns pontos dele para que a ferramenta fosse melhor moldada às novas necessidades que chegavam a todo instante. Dessa forma, a interface final da aplicação chegou ao ponto atual.

4.2.3 Estrutura da aplicação

A organização do código é dividida basicamente em quatro partes:

View ⇒ Cuida da interface com o usuário;

Core ⇒ Base da aplicação;

Callback ⇒ Biblioteca de funções;

Integração ⇒ Integração entre a interface e a base da aplicação.

Esta estrutura foi baseada no padrão de arquitetura de software bastante conhecida por MVC, que é o acrônimo de *Model-View-Controller*.

A sessão *View*, possui um dos padrões de projeto, como *factory*, para criação de componentes para interface.

A sessão *Integração*, possui uma variação do padrão de projeto *observer* [9], para a conversação entre as camadas *View* e Core.

O *Core* cuida principalmente do manuseio do *GStreamer*. A estrutura interna é descrita como na figura abaixo:

Da necessidade de se executar mais de um vídeo ao mesmo tempo, adicionou-se um novo requisito ao projeto, que é o sincronismo. Isso significa que, se no início da apresentação o vídeo A se encontrar no frame x, e o vídeo B no y, com x, $y \in \mathbb{N}$; durante toda exibição, a diferença de frames entre A e B será x - y.

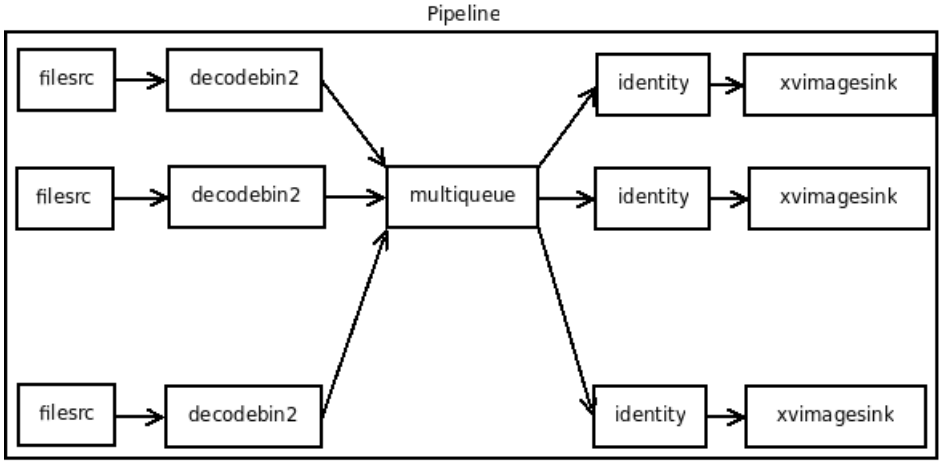


Figura 4.11: Estrutura dos elementos do gstreamer

Da figura acima é possível observar 5 elementos distintos:

- **filesrc:** elemento responsável pela leitura de dados de um arquivo;
- **decodebin2:** designado para que detecte o tipo de mídia no *stream* de entrada e, através dos *decoders* disponíveis, utilizar o adequado (se este existir no sistema);
- **multiqueue:** incumbido de cuidar da concorrência de múltiplos *streams* de dados;
- **identity:** elemento "identidade", i.e. os dados que este recebe são enviados por ele sem modificações. Mas possui um papel vital na aplicação, pois é por onde as *queries* sobre o *stream* de dados são feitas;
- **xvimagesink:** encarregado de exibir os dados recebidos por ele em uma saída, no caso a tela.

4.2.4 Starvation

Cada vídeo executado é um processo competindo por recursos do processador. Esta concorrência dos processos faz com que haja a necessidade de políticas de escalonamento, para que seja decidido quem receberá um determinado recurso. Estas políticas podem acarretar em problemas, como um processo nunca vir a receber um recurso. Este tipo de ocorrência, na área de *Programação Concorrente*, é conhecido por **starvation**.

Para solucionar este problema foi utilizado um elemento do *GStreamer*, o *multiqueue*. Este, para assegurar que nenhum processo sofra de *starvation*, faz uso de filas que podem crescer dinamicamente (até um certo limite), garantindo a qualquer momento que ao menos N *bytes* de cada um deles esteja na fila [1].

4.3 Licença

O *Visualizador de Múltiplos Vídeos* poderá ser distribuído sob as condições da *GNU General Public License*(GPL).

Capítulo 5

Resultados Obtidos

Apesar de termos nos deparado com alguns problemas e limitações, os quais estão descritos abaixo, o aplicativo pareceu corresponder a grande parte das expectativas durante sua avaliação. Embora algumas funcionalidades adicionais acabassem não sendo desenvolvidas, todas as *features* inicialmente propostas no projeto cumpriram seus objetivos. Estas características seriam:

- Exibição simultânea de vídeos;
- Possibilidade de sincronizar manualmente os vídeos;
- Implementação de uma interface com usabilidade.

5.1 GStreamer

Para versões abaixo de 0.10.25 do GStreamer, foi percebida uma falha na implementação de um método relativamente importante para o projeto (*gst.Element.query_ duration()* [4]), o que nos forçou a utilizar o *framework* apenas na versão citada, ou outra superior.

5.2 GTK

Versões do arcabouço GTK igual a 2.18 e do *wrapper* PyGTK 2.16 sofreram mudanças em alguns métodos, alterando o comportamento da funcionalidade de seleção do quadrante em que um determinado vídeo está tocando. Então, por enquanto a versão mínima para execução do aplicativo com a *feature* funcionando corretamente é 2.14.4 para GTK e 2.13.0 para o PyGTK, e menor que 2.18 e 2.16 respectivamente para GTK e PyGTK.

5.3 Falta de precisão

Na procura por um frame em específico, ocorre uma falta de precisão para obtê-lo. Acreditamos que esse problema seja causado pela utilização de alguns tipos de algoritmos de compressão no arquivo de vídeo.

Por exemplo, um dos tipos de compressão mais utilizados é o *interframe compression*. Ele tira proveito do fato de que grande parte dos vídeos possuem mudanças bastante sutis de um

frame para outro, fazendo com que durante a compressão sejam armazenadas somente mudanças incrementais entre esses quadros, exceto quando existirem alterações drásticas na imagem (e.g. troca entre uma cena e outra), forçando à criação de um quadro novo no qual a imagem inteira é guardada, sendo isto chamado de *key frame*. Este último também pode sofrer compressão, mas nesse caso são utilizados algoritmos de compactação na imagem. Este método é chamado de *intraframe compression*.

Pelo fato de os frames representarem o quão diferente eles são em relação aos anteriores, não é possível avançar ou retroceder para um quadro em específico no vídeo. Por isso, alguns compressores criam um *key frame* a cada intervalo de tempo, para que seja possível ir rapidamente para um quadro próximo ao qual se queira buscar.

Alguns formatos foram testados para diversos *codecs* diferentes. O intervalo de quadros entre *key-frames* varia de vídeo para vídeo. A seguinte tabela mostra uma média dos resultados obtidos a partir da avaliação de alguns vídeos avi:

Codecs para AVI				
Codec	mpeg1	mpeg4	x264	xvid
Distância entre Key-Frames [a]	11.2	6.3	7.6	7.2

[a] Em frames

5.4 Erros encontrados

Para alguns tipos de arquivos testados, atributos tais como o quadro atual e o total não se mostravam disponíveis, refletindo no mau funcionamento da *trackbar* e das informações mostradas no canto inferior da ferramenta. Assim também não era possível avançar ou retroceder.

A seguinte tabela indica alguns dos formatos que são ou não suportados pela ferramenta atualmente:

Formatos Testados								
Formato	ogg	avi	mov	mp4	mpg	wmv	mkv	flv
Funcionando	✓	✓	×	×	×	×	×	×

Capítulo 6

Conclusão

Apesar dos problemas retratados em **Resultados Obtidos** 5, no geral a ferramenta se mostra de grande valia. A maior parte dos erros não tratados não chega a ser um problema para o usuário. O único que pode ser considerado realmente sério é o da imprecisão no buscador de *frames*. Para o futuro deste projeto, seria interessante que esse problema fosse corrigido.

Falando em futuro do projeto, muitos adicionais que havíamos pensado não puderam ser implementados. Logo, aqueles que quiserem dar continuidade ao trabalho não terão problemas em encontrar algo para fazer: implementar uma tela para a sincronização dos vídeos, que permita ao usuário colocar o vídeo na posição inicial desejada sem a necessidade de saber de antemão essa informação; a possibilidade de aplicar o processamento em um vídeo ao mesmo tempo em que a aplicação o reproduz; otimização das *threads* do programa, oferecendo a oportunidade de visualizar ainda mais vídeos; visualização de outros tipos de mídia, como fotos ou *slides*.

Até onde se sabe, atualmente não existe nenhuma aplicação famosa semelhante à desenvolvida neste trabalho. Isso adicionou ao projeto uma responsabilidade muito grande, pois ele pode vir a ser usado por todo um nicho da comunidade computacional. Assim sendo, a ferramenta não deveria ser apenas funcional, mas também usável. Neste ponto, ela parece cumprir o seu papel, pois os testes realizados indicam que a interface implementada é bastante intuitiva, e minimalista o suficiente para que o usuário possa fazer o que precisa sem ficar confuso no processo.

Esperamos que, com este trabalho, tenhamos conseguido atingir a meta de auxiliar os usuários de visão computacional, trazendo agilidade e praticidade para suas tarefas.

II. Parte Subjetiva

Capítulo 7

Desafios, Frustrações e Planos Futuros (Edson)

7.1 Desafios e Frustrações

Um dos desafios no projeto foi conciliar o trabalho de formatura supervisionado com as matérias que cursei durante o período. E também a implementação de uma estrutura de elementos do *GStreamer* na qual fosse possível a exibição de múltiplos vídeos concorrentemente além de manter a sincronia entre eles.

Embora tenha conseguido superar os desafios, não foi possível concluir completamente tudo o que havia planejado inicialmente, como a correção de alguns *bugs* e a adição de algumas outras funcionalidades.

Outra frustração foi o fato de a ferramenta não poder funcionar para alguns tipos de vídeo, devido à estrutura do arquivo ou algoritmos de compressão utilizados nele.

7.2 Disciplinas

Lista das disciplinas cursadas que se mostraram possuir maior relevância para a realização deste trabalho:

- **MAC0110 - Introdução à Computação:** pelo fato de que nunca havia programado antes de entrar no IME;
- **MAC0122 - Princípios de Desenvolvimento de Algoritmos:** por ser uma base importante para escrever algoritmos com maior complexidade;
- **MAC0323 - Estrutura de Dados:** importante aprendizado de manipulação e estudo de estruturas de dados;
- **MAC0332 - Engenharia de Software:** conhecimento do processo e gerenciamento de desenvolvimento de softwares;
- **MAC0441 - Programação Orientada a Objetos:** pelos fundamentos de POO e padrões de desenvolvimento de software ;

- **MAC0446 - Princípios de Interação Homem-Computador:** alicerce para o desenvolvimento de uma interface amigável e intuitiva da ferramenta;
- **MAC0438 - Programação Concorrente:** pelos fundamentos de programação concorrente, dentre eles a sincronização entre processos;
- **MAC0417 - Visão e Processamento de Imagens:** conceitos importantes para o entendimento dos fundamentos do projeto.

7.3 Planos Futuros

Um objetivo que posteriormente poderia ser alcançado, seria a utilização de algoritmos de processamento de imagem ao mesmo tempo em que o vídeo estaria sendo exibido, sendo que os resultados das transformações poderiam ser apresentadas ao lado da original ou de outros processamentos para efeito de comparação. Dessa forma, o usuário não teria a necessidade de processar o vídeo de antemão e armazená-lo num arquivo, fazendo com que os *frames* processados possam correr o risco de sofrer alterações indesejáveis devido ao tipo de compressão que o usuário decidir utilizar.

Outro objetivo seria a estruturação da aplicação de forma que estes algoritmos possam ser facilmente intercambiáveis além da possibilidade de adição de novos algoritmos pelo usuário.

Capítulo 8

Desafios, Frustrações e Planos Futuros (Marum)

8.1 Desafios e Frustrações

Durante boa parte do desenvolvimento deste trabalho eu também estava fazendo estágio (início em maio de 2009). Isso significa que 30 horas por semana não poderiam ser dedicadas ao projeto. Somando a isso as disciplinas que eu também estava cursando e os trabalhos referentes a elas, restava apenas uma pequena parcela do meu tempo para efetivamente trabalhar em cima dele. Uma enorme capacidade de organização foi necessária para que tudo isso fosse possível.

Trabalhar com uma linguagem que nunca tínhamos visto antes também foi um desafio. Felizmente, nós conseguimos nos adaptar a ela rapidamente. Mais do que isso, a agilidade no desenvolvimento oferecida pelo Python foi essencial para que pudéssemos conceber o *Visualizador de Múltiplos Vídeos*. Ouso dizer aqui que se o projeto tivesse sido desenvolvido em qualquer uma das linguagens que já conhecíamos, talvez ele não tivesse sido concluído a tempo.

Entender e manipular a estrutura do *GStreamer* também se mostrou mais complexo do que imaginávamos. Acreditamos que a documentação desta ferramenta ainda está um pouco pobre, e peca quando você procura por exemplos, ou sobre os seus recursos.

Finalmente, os problemas com ambiente eram frequentes. Mais do que um incômodo, por diversas vezes eles atrasaram e muito o desenvolvimento do trabalho. A distribuição do linux que utilizávamos, bem como a versão de alguma biblioteca, sempre afetavam o nosso projeto de alguma forma, seja pela apresentação dele para o usuário, ou até mesmo comprometendo-o completamente.

Gostaria também de acrescentar uma insatisfação a respeito do trabalho concluído, que infelizmente ainda possui algumas limitações, alguns bugs que não conseguimos corrigir, e também não conta com alguns extras que não puderam ser implementados.

8.2 Disciplinas

Foram muitas as disciplinas relevantes no desenvolvimento do trabalho. Em maior ou menor grau, todas tiveram alguma participação. Segue aqui uma lista das que julgo terem sido as mais importantes:

- **MAC0110 - Introdução à Computação:** antes de me ingressar em ciência da computação aqui no IME, eu nunca tinha programado nada. Esta foi a disciplina que me iniciou neste mundo, e uma boa base nela foi imprescindível ao longo do curso;
- **MAC0122 - Princípios de Desenvolvimento de Algoritmos:** nesta matéria aprendemos a criar algoritmos mais complexos, e pela primeira vez somos confrontados com problemas, como a eficiência de um programa segundo o custo de tempo e memória. Para este trabalho, foi importante otimizar o uso da memória, a fim de conseguir reproduzir o maior número de vídeos possível;
- **MAC0211 - Laboratório de Programação I:** aqui tivemos o primeiro contato com um projeto de maior porte, que também era o primeiro em grupo;
- **MAC0332 - Engenharia de Software:** aprendemos algumas formas de gerenciar o desenvolvimento de um software. Organização, distribuição de papéis e padronização do código são alguns dos conceitos utilizados durante a implementação da ferramenta;
- **MAC0342 - Laboratório de Programação Extrema:** algumas das práticas desta metodologia foram utilizados para a realização de nosso trabalho;
- **MAC0441 - Programação Orientada a Objetos:** diversos conceitos importantes foram ensinados nesta disciplina. Para o *Visualizador de Múltiplos Vídeos*, utilizamos alguns dos padrões de projeto vistos em aula;
- **MAC0446 - Princípios de Interação Homem-computador:** essa disciplina prega que de nada vale o sistema se o usuário não tiver prazer em usá-lo. Ou seja, a interface não pode ser desprezada durante o desenvolvimento do seu software e, para o nosso trabalho, não poderia ser diferente;
- **MAC0438 - Programação Concorrente:** a sincronização era fundamental em nosso trabalho, e por isso os conceitos abordados nessa matéria se mostraram muito válidos;
- **MAC0417 - Visão e Processamento de Imagens:** apesar de eu não ter cursado esta disciplina, precisei me interar com ela para este trabalho. Afinal, se o nosso software seria utilizado pelo pessoal da visão computacional, o mínimo que eu deveria fazer era conhecer um pouco sobre esse mundo.

8.3 Planos Futuros

Acredito que a primeira coisa que faria se continuasse atuando no desenvolvimento deste trabalho seria corrigir todas as limitações e problemas encontrados, para só então começar a atuar em cima de novas funcionalidades.

Entre estas novas funcionalidades, gostaria que uma tela para sincronização fosse implementada, evitando assim que o usuário precisasse saber de antemão em qual frame o vídeo deve ser inicializado. Outra funcionalidade que pensamos para a ferramenta que gostaria de ver implementada seria a possibilidade de aplicar o processamento durante a exibição do vídeo.

Referências Bibliográficas

[1] *GStreamer Core Plugins 0.10 Plugins Reference Manual - Multiqueue.* http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer-plugins/html/gstreamer-plugins-multiqueue.html.

[2] *GStreamer: Open Source Multimedia Framework.* http://gstreamer.freedesktop.org.

[3] *The GTK+ Project.* http://www.gtk.org.

[4] *The Python GStreamer Class Reference - gst.Element.* http://pygstdocs.berlios.de/pygst-reference/class-gstelement.html#method-gstelement--query-duration.

[5] *Python Programming Language – Official Website.* http://www.python.org/.

[6] Dana H. Ballard, Christopher M. Brown: *Computer Vision.* Prentice-Hall, Inc, Printed in United States of America, 1982, ISBN 0-13-165316-4.

[7] Dave Brueck, Stephen Tanner: *Python 2.1 Bible.* Hungry Minds, Inc., Printed in United States of America, 2001, ISBN 0-7645-4807-7.

[8] Hetland, Magnus Lie: *Beginning Python From Novice to Professional.* APress, Printed in United States of America, 1982, ISBN 1-59059-519-X.

[9] Holzner, Steve: *Design Patterns For Dummies.* Wiley Publishing, Inc., Manifactured in United States of America, 2006, ISBN 0-471-79854-1.

[10] Ian T. Young, Jan J. Gerbrands: *Fundamentals of Image Processing.* Printed in Netherlands at the Delft University of Technology, 1998, ISBN 90-75691-01-7.

[11] Kniberg, Henrik: *Scrum and XP from the Trenches.* 2007, ISBN 978-1-4303-2264-1.