Um Player e Uma Estratégia de Sincronização de Vídeo Distribuído

Caio Marcelo Campoy Guedes

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2018

Caio Marcelo Campoy Guedes

Um Player e Uma Estratégia de Sincronização de Vídeo Distribuído

Monografia apresentada ao curso Ciência da Computação
do Centro de Informática, da Universidade Federal da Paraíba,
como requisito para a obtenção do grau de Bacharel em titulo

Orientador: Lincoln David Nery e Silva

Novembro de 2018

Catalogação na publicação
Seção de Catalogação e Classificação

G924p Guedes, Caio Marcelo Campoy.
Um Player e Uma Estratégia de Sincronização de Vídeo Distribuído / Caio Marcelo Campoy Guedes. - João Pessoa, 2018.
42 f.

Orientação: Lincoln David Nery e Silva.
Monografia (Graduação) - UFPB/CI.

1. Multimídia. 2. Player de Vídeo. 3. Arquitetura Distribuída. 4. Sincronização de Vídeo. I. Silva, Lincoln David Nery e. II. Título.

UFPB/BC

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

Trabalho de Conclusão de Curso de Ciência da Computação intitulado ***Um Player e Uma Estratégia de Sincronização de Vídeo Distribuído*** de autoria de Caio Marcelo Campoy Guedes, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Dr. Lincoln David Nery e Silva
Universidade Federal da Paraíba

Prof. Dr. Guido Lemos de Souza Filho
Universidade Federal da Paraíba

Prof. Dr. Carlos Eduardo Coelho Freire Batista
Universidade Federal da Paraíba

João Pessoa, 12 de Novembro de 2018

Centro de Informática, Universidade Federal da Paraíba
Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600
Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

"We make our world significant by the courage of our questions and the depth of our answers."

Carl Sagan

Dedico este trabalho a minha familia, aos meus amigos e aos professores que me auxiliaram nessa caminhada.

AGRADECIMENTOS

Agradeço primeiramente a minha família, meu pai Marcelo, minha mãe Cleide e meu irmão Vitor, por terem me incentivado e auxiliado durante essa trajetória. Agradeço todo o esforço e dedicação que em mim foram investidos.

Agradeço aos grandes amigos que fiz durante a minha passagem no curso e aos professores e pesquisadores do Núcleo de Pesquisa e Extensão *LAViD* (Laboratório de Aplicações de Vídeo Digital) por todo o conhecimento que me foi dado.

Finalizo agradeçendo ao meu orientador e aos professores responsáveis pela banca, pelas críticas construtivas durante a elaboração desse trabalho.

RESUMO

Com a evolução e surgimento de novos padrões de vídeo de ultra alta definição, novos modelos de reprodução da mídia vem sendo desenvolvidos tanto em *hardware* quanto em *software*. O custo atrelado a estas tecnologias é elevado, fazendo com que o acesso a estas seja limitado a um grupo seleto. Neste trabalho utilizamos uma solução de vídeo distribuído que faz uso de múltiplos computadores de baixo custo e desenvolvemos uma heurística utilizando *visão computacional* para garantir a sincronização do conteúdo.

Palavras-chave: <Multimídia>, <Player de Vídeo>, <Arquitetura Distribuída>, <Síncronização de Vídeo>.

ABSTRACT

With the evolution and emergence of new ultra high definition video standards, new media playback models have been developed in both hardware and software. The cost associated with these technologies is high, making access limited to a select group. In this work we use a distributed video solution that makes use of multiple low cost computers, to guarantee the synchronization of the content a heuristic that uses computer vision has been developed.

Key-words: <Multimedia>, <Video Player>, <Distributed Architecture>, <Video Synchronization>.

LISTA DE FIGURAS

1 Exemplificação da passagem da luz até o sensor de uma câmera. Fonte [2]. 19

2 Exemplo do comportamento da luz com relação a frequência e o comprimento de onda. https://goo.gl/Hqu1qT. Acessado em: Novembro de 2018. 20

3 Exemplificação da ordem de exibição dos quadros de um vídeo com relação ao tempo. O vídeo inicia no tempo 0 junto ao primeiro frame e se desloca até um tempo *n*. 21

4 Exemplo do funcionamento dos vetores de movimento. As setas brancas indicam o deslocamento na cena. https://goo.gl/LHsUQF. Acessado em: novembro de 2018 22

5 Exemplo do posicionamento e relação entre os quadros I, B e P na *timeline* de um vídeo. https://goo.gl/VAVPG2. Acessado em: novembro de 2018. . 23

6 Exemplo da mudança de informação entre dois quadros. A imagem da esquerda representa o quadro I, a da direita representa o quadro B, os retângulos cinzas indicam áreas que devem ser extraídas do quadro I e posicionadas no B, pois não sofreram alteração. https://goo.gl/igPhs2. Acessado em: novembro de 2018. 24

7 Canais R, G e B extraídos de uma imagem colorida. https://goo.gl/2dujqv. Acessado em: novembro de 2018. 25

8 Canais Y, U e V extraídos de uma imagem colorida. https://goo.gl/7mcR4B. Acessado em: novembro de 2018. 25

9 Exemplificação dos padrões do QR code. https://goo.gl/rEe95u. Acessado em: novembro de 2018. 26

10 Comparação entre diferentes tipos de resolução de vídeo. https://goo.gl/7B8Nb2. Acessado em: Novembro de 2018 28

11 Representação da designação da função de cada computador que realiza a exibição. 29

12 Exemplificação de como os QR codes são exibidos na telas dos players. . . 30

13 Extração e envio da trilha de vídeo ao decodificador. 30

14 Exemplificação do funcionamento da fila circular. 32

15 Representação da exibição do conteúdo com e sem o preenchimento de bordas 33

16 Exemplificação da numeração das telas. 34

17 Arquitetura da solução proposta englobando observador e players. 35

18 Telas dessincronizadas. A tela 1 apresenta quadros a frente da referência e as telas 2 e 3 apresentam quadros prévios a referência. 37

19 Exemplificação da captura da câmera durante a atualização do monitor. . . 38

20 Telas durante o processo de sincronia. Quadrantes com o texto avermelhado descartam quadros, os com o texto esverdeados aguardam durante um intervalo de tempo. 39

21 QR codes capturados durante a atualização das telas. 39

22 Telas sincronizadas com a solução de correção de bordas. 40

23 Telas sincronizadas com a solução de correção de bordas em um ponto futuro. 40

LISTA DE TABELAS

1 Comparação do *bitrate* de vídeos com resolução e *frame rate* distintos. . . . 22
2 Ações e comandos suportados pelo *player* 35
3 Informação dos vídeos utilizados para a validação da solução 36
4 Configuração das máquinas utilizadas para executar os players 36
5 Configuração da máquinas utilizada para executar o observador 36

LISTA DE ABREVIATURAS

TCP – Transmission Control Protocol

QR – Quick Response

CCD – Charge-Coupled Device

CMOS – Complementary Metal-Oxide-Semiconductor

DTS – Decode Time Stamp

PTS – Presentation Time Stamp

JPEG – Joint Photographic Experts Group

LIBRAS – Língua Brasileira de Sinais

RGB – Red-Green-Blue

AIM – Advancing Identification Matters

JIS – Japanese Industrial Standard

ISO – International Organization for Standardization

UHD – Ultra High Definition

HEVC – High Efficiency Video Coding

SAGE – Scalable Amplified Group Environment

GPU – Graphics Processing Unit

UDP – User Datagram Protocol

TCP – Transmission Control Protocol

HTTP – Hypertext Transfer Protocol

RTP – Real-time Transport Protocol

API – Application Programming Interface

VDPAU – Video Decode and Presentation API for Unix

VAAPI – Video Acceleration API

SDL – Simple DirectMedia Layer

Conteúdo

1 INTRODUÇÃO **17**

1.1 Premissas e Hipóteses 17

1.1.1 Objetivo geral 18

1.1.2 Objetivos específicos 18

1.2 Estrutura da monografia 18

2 FUNDAMENTAÇÃO TEÓRICA **19**

2.1 Vídeo Digital 19

2.1.1 Linha do Tempo 20

2.1.2 Compressão e Decodificação 21

2.1.3 Exibição dos frames 24

2.1.4 *QR Code* 25

3 TRABALHOS RELACIONADOS **27**

4 SOLUÇÃO PROPOSTA **28**

4.1 ARQUITETURA 29

4.2 *PLAYER* 29

4.3 OBSERVADOR 33

5 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS **36**

6 CONCLUSÕES E TRABALHOS FUTUROS **41**

REFERÊNCIAS **41**

1 INTRODUÇÃO

Novos formatos de compressão de vídeo e a disponibilidade de equipamentos mais sofisticados permitem a evolução e manipulação de resoluções cada vez mais altas. Quanto maior a quantidade de pixels adicionados ao receptor de uma câmera, maior a fidedignidade da imagem capturada, maior será a imersão, o ângulo de visão e o nível de detalhes e, dessa forma, melhor será a sua qualidade[1].

Embora o 4K ainda esteja no estado da arte, já é possível perceber a chegada do 8K no mercado. Serviços de *streaming* como YouTube[1] já fornecem mídias desse porte, porém a capacidade de reproduzi-las depende da largura de banda, do poder computacional do dispositivo reprodutor e da presença de uma tela adequada. Em seu trabalho de 2014, Becker[1] afirma que embora o 4K seja a referência mundial, no Brasil, o uso desse tipo de experiência se faz restrito a aplicações artísticas e de telessaúde. A passagem do autor está defasada, pois esse tipo de mídia está muito mais acessível. Entretanto, é possível considerar a premissa verdadeira para o novo padrão de mercado, que é o 8K. Para que o acesso a esse tipo de mídia possa ser difundido, pode-se utilizar uma estratégia de divisão e conquista onde diferentes máquinas trabalham em conjunto para exibir um conteúdo de larga escala.

A sincronização da mídia em um cenário distribuído em que múltiplas máquinas atuam para a reprodução de partes de um mesmo conteúdo é um grande desafio. Além de relógios diferentes, tais computadores podem possuir hardware distintos. Isso significa que o conteúdo ao ser reproduzido pode apresentar quadros diferentes. Portanto, constante refinamento da sincronia entre os computadores se faz necessário.

Ao lidar com uma abordagem distribuída, espera-se uma solução escalável, já que ao aumentar a quantidade de máquinas efetuando o trabalho de decodificação e exibição da mídia pode-se criar um player capaz de reproduzir mídias de dimensão nK. Esse sistema pode ser utilizado em ambientes educacionais onde a riqueza de detalhes e a dimensão da imagem são cruciais para a compreensão do que está sendo lecionado. A aplicação na medicina, por exemplo, pode auxiliar no entendimento das particularidades do corpo humano.

1.1 Premissas e Hipóteses

Através da observação de um player distribuído foi possível constatar que se a informação do *frame* atual de cada tela for informada a um observador, há a possibilidade de realizar a sincronia dos fragmentos do vídeo. O observador deve ser capaz de analisar a discrepância entre os *frames* e manipular os *players* indicando a quantidade de atraso ou

[1]https://www.youtube.com

adiantamento que os mesmos devem efetuar para que os quadros que estão sendo exibidos sejam os mesmos em todas as telas.

1.1.1 Objetivo geral

Este trabalho estuda a viabilidade da utilização de uma estratégia em um player de vídeo distribuído que faz uso de *QR codes* para manter a sincronização na exibição de n partes.

1.1.2 Objetivos específicos

- Investigar soluções existentes que permitem a sincronização de vídeo distribuído;
- Implementar um *player* com suporte a aceleração de *hardware*;
- Exibir *QR code* com informações do *frame* atual no *player*;
- Permitir o controle de atraso ou avanço da reprodução no *player* por mensagens de controle;
- Implementar um observador que realiza a sincronização dos segmentos;
- Avaliar o desempenho da solução proposta para mídias de diferentes resoluções e quadros por segundo.

1.2 Estrutura da monografia

Na segunda seção deste trabalho serão introduzidos os fundamentos para que seja possível a sua compreensão. Na terceira seção serão apresentados trabalhos relacionados que lidam com a reprodução de vídeo em um cenário distribuído. Na quarta seção são apresentadas as funcionalidades da solução proposta bem como as suas aplicações. Na quinta seção serão discutidos os detalhes da implementação e dos casos de teste utilizados para a avaliação do sistema. Na última seção discutiremos trabalhos a serem realizados para adaptar a solução proposta a diferentes cenários de uso assim como a conclusão do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentados com suporte da literatura, os elementos teóricos que suportam este trabalho.

2.1 Vídeo Digital

O uso do vídeo tem se aprimorado desde o século passado. Em 1940, a mídia começou a ser explorada através da criação da televisão. Desde então, foi possível presenciar diversos fatos históricos através desta mídia[2]. Atualmente, a quantidade de tecnologias se multiplicou, o acesso a vários tipos de conteúdos a partir de computadores utilizando serviços de *streaming* e a utilização de celulares para visualização de gravações são alguns exemplos do aumento de consumo desses recursos. Os vídeos surgiram em sua forma analógica, câmeras filmadoras permitem a passagem da luz por um curto intervalo de tempo sobre um rolo. Em seguida, a reação química da luz com o filme que está presente no rolo cria padrões sobre ele e assim temos uma imagem gravada. Atualmente, utiliza-se o meio digital para capturar os vídeos. O princípio da captura é o mesmo: um obturador permite a passagem da luz sobre um curto intervalo de tempo. Entretanto, ao invés do contato da luz com um fragmento do rolo, tem-se o contato com um chip. A Figura 1 exemplifica a chegada da luz ao sensor. Chips como o CCD e o CMOS são formados por milhões de pequenos capacitores espalhados por uma superfície. Quando a câmera permite a passagem da luz, eles são bombardeados por fótons (partículas elementares que carregam energia). Em seguida, a câmera realiza a análise da intensidade de carga armazenada nos capacitores e então uma imagem digital é formada. Cada capacitor quantiza a carga com base em um segmento do espectro de luz visível. Em geral, divide-se o chip em um mosaico de capacitores individuais que capturam as porções verde, vermelho e azul. A Figura 2 ilustra o espectro da luz. Os chips são mensurados pela quantidade de píxeis que possuem em seu sensor e são denominados de 'Megapixels'. Dessa forma, quanto maior a quantidade de capacitores em um chip maior será a qualidade da imagem.

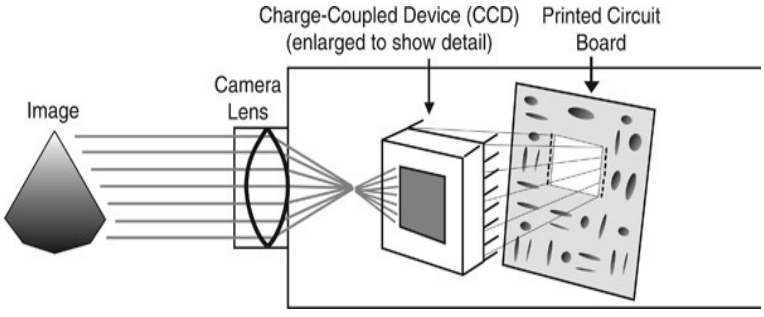

Figura 1: Exemplificação da passagem da luz até o sensor de uma câmera. Fonte [2].

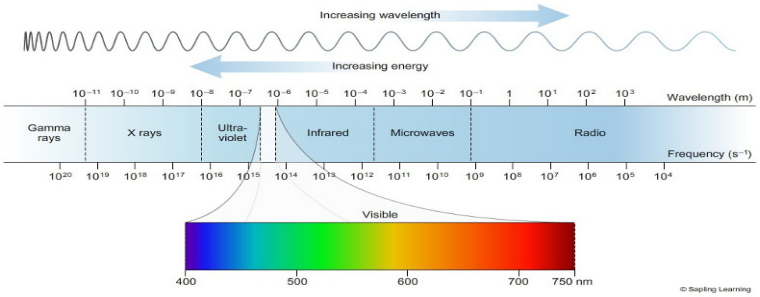

Figura 2: Exemplo do comportamento da luz com relação a frequência e o comprimento de onda. https://goo.gl/Hqu1qT. Acessado em: Novembro de 2018.

2.1.1 Linha do Tempo

Segundo Tekalp[3], vídeos digitais são uma sequência 2D ou 3D de imagens com ou sem movimento. Entretanto, uma imagem é constante no que diz respeito ao tempo. Tekalp afirma que vídeos são sequências de imagens que seguem um padrão espaço-temporal de intensidade que varia com o andamento do tempo. Ou seja, vídeos são mídias 3D ou 4D, onde além das dimensões de largura, altura e, em alguns casos, profundidade, também possuem uma dimensão extra, que é a temporal.

A timeline é a visão dos *frames* de um vídeo com relação ao tempo, conforme ilustra a Figura 3. Os vídeos são gravados com base no *frame rate*, que significa a amostragem de frames em um segundo. De acordo com Read[4], o olho humano é capaz de transmitir entre 10 e 12 imagens em um segundo ao cérebro. Desse modo, uma imagem a cada 1/15 segundo. Logo, se uma variação da imagem atinge o olho nesse intervalo de tempo, tem-se a sensação de continuidade.

Na era do cinema mudo, a taxa mínima estipulada pela indústria cinematográfica iniciou-se em 16 fps (*frames per second*). Atualmente, a taxa base é de 24 fps para filmes, sendo possível a exibição de conteúdos com taxas superiores a 60 fps. Para que o vídeo siga a sua *timeline* de forma fiel, deve-se controlar a exibição com marcações nos quadros que indicam quando determinado *frame* deve ser decodificado e quando ele deve ser exibido. Essas marcações são chamadas respectivamente de DTS e PTS.

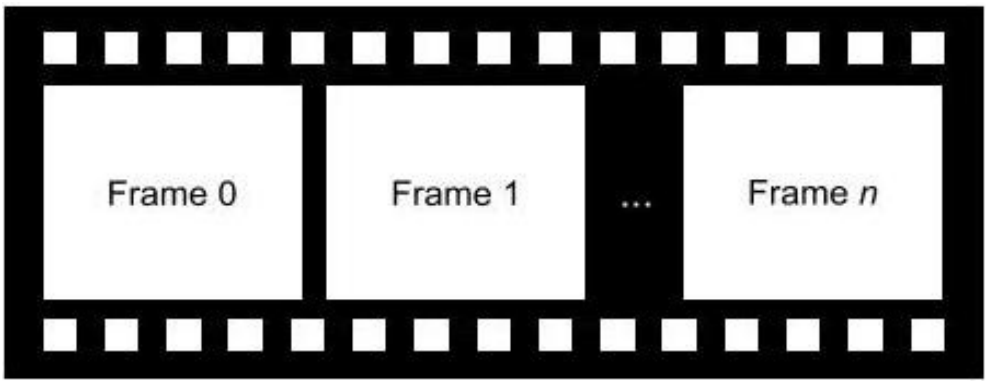

Figura 3: Exemplificação da ordem de exibição dos quadros de um vídeo com relação ao tempo. O vídeo inicia no tempo 0 junto ao primeiro frame e se desloca até um tempo *n*.

O DTS de um vídeo organiza qual quadro deve ser decodificado primeiro, existem tipos de frames diferentes, são eles, P, B e I, a explicação detalhada do que cada frame representa será feita na seção 2.1.2, nesse instante o que se faz necessário compreender é que a ordem desses frames é importante para o decodificador. Portanto, o DTS é responsável por informar ao decodificador qual a ordem correta em que os quadros devem ser decodificados baseado em um *timestamp*.

O PTS é uma referência utilizada para exibir cada quadro em seu tempo correto. Um *clock* base é utilizado pelo programa que pretende reproduzir o vídeo, este acompanhado do PTS mantém a reprodução da mídia em um tempo apropriado.

2.1.2 Compressão e Decodificação

A qualidade do vídeo varia conforme o conteúdo é digitalizado, a amostragem utiliza a luminância(Y) e a crominância(Cr e Cb) de uma imagem, já a quantização está relacionada à quantidade de *frames* expostos ao chip da câmera em um intervalo de um segundo, o qual é conhecido como fps[5]. O bitrate de um vídeo é a combinação da magnitude em bits da amostragem pela quantização, ou seja, quanto maior for o tamanho da imagem e o fps, mais informação teremos em um vídeo. A Tabela 1 compara vídeos de dimensões e *frame rate* diferentes e mesma profundidade de cor(24 bits, ou seja, um byte para cada canal RGB) sem compressão, o resultado é o *bitrate* do vídeo dado em Gbps (*Gigabits* por segundo). O cálculo do *bitrate* em *bytes* por segundo é dado por:

$$\sum_{0}^{r} W \times H \times D$$

Onde r é o *frame rate* do vídeo, W a sua largura, H a sua altura e D a profundidade de cor.

Largura (*Pixels*)	**Altura(*Pixels*)**	**Escaneamento**	***Frame rate***	***Bitrate***
1920	1080	Progressivo	24	1.19 Gbps
1920	1080	Progressivo	60	2.99 Gbps
3840	2160	Progressivo	24	4.79 Gbps
7680	4320	Progressivo	24	19.1 Gbps

Tabela 1: Comparação do *bitrate* de vídeos com resolução e *frame rate* distintos.

Como pode ser inferido da Tabela 1, os padrões atuais demandam uma quantidade imensa de armazenamento. Logo, técnicas de compressão precisaram ser elaboradas. Com a ideia de que o vídeo é uma sucessão de imagens, pode-se a princípio utilizar o padrão de compressão de imagens JPEG em cada uma das imagens do filme. Esse tipo de compressão é intitulada MJPEG (Moving JPEG). Entretanto, esse tipo de compressão utilizando JPEG não é significativa[5]. Além da redundância espacial presente nos frames, também deve ser levada em consideração a redundância entre eles. Durante uma gravação, a probabilidade de que o conteúdo de uma cena mude repentinamente é muito pequena. Assim, os algoritmos de compressão devem levar em consideração a possibilidade de remover essas seções redundantes. Em uma tomada de um intérprete de LIBRAS, por exemplo, pode-se detectar mudanças no tronco, na face e nas mãos do indivíduo, então um vetor de movimento pode ser calculado, indicando que segmentos do *frame* anterior devem ser redesenhados. A Figura 4 ilustra os vetores de movimento em uma cena do filme Sintel da Blender Foundation.

Figura 4: Exemplo do funcionamento dos vetores de movimento. As setas brancas indicam o deslocamento na cena. https://goo.gl/LHsUQF. Acessado em: novembro de 2018

Os *frames* podem ser divididos em três tipos majoritários:

- *Frames I*: Denominados *intracoded frames*, que são quadros que representam uma imagem completa;

- *Frames P*: Denominados *predictive frames*, que são quadros que precisam de um *frame* I predecessor para formar uma imagem;

- *Frames B*: Denominados *bidirectional frames*, que são quadros que utilizam *frame* I e P para formar a imagem, estes são situados antes ou após a sua presença.

A Figura 5 exemplifica como os *frames* I, B e P relacionam-se em uma *timeline*(t). Eles são criados pelo compressor, que extrai o máximo de informação redundante dos quadros e vincula a eles um *timestamp*, denominado DTS. A Figura 6 ilustra dois frames, em que o da esquerda representa um *frame* I e o da esquerda representando um *frame* B ou P. Os blocos pintados de cinza são informações redundantes que foram descartadas pelo compressor. Vale ressaltar que não são todos os meios de compressão que utilizam esses tipos de *frames*. Contudo, a grande maioria o faz, embora alguns podem não utilizar *frames* B ou podem usar um *frame* chamado de D, cujos detalhes não são necessários para as explicações vindouras.

O trabalho do decodificador é extrair o *frame* com base nas informações que são enviadas junto ao cabeçalho do *codec*, que são: altura, largura, *pixel format*, *frame rate*, dentre outras. A outra responsabilidade é decodificar a mídia em seu tempo correto utilizando as informações de DTS. A Figura 5 ilustra a importância da ordem de decodificação, para os primeiros 4 quadros, que, de acordo com a mesma são, IBBP, cuja ordem deveria ser IPBB, já que o *frame* P depende do I e os *frames* B dependem de ambos.

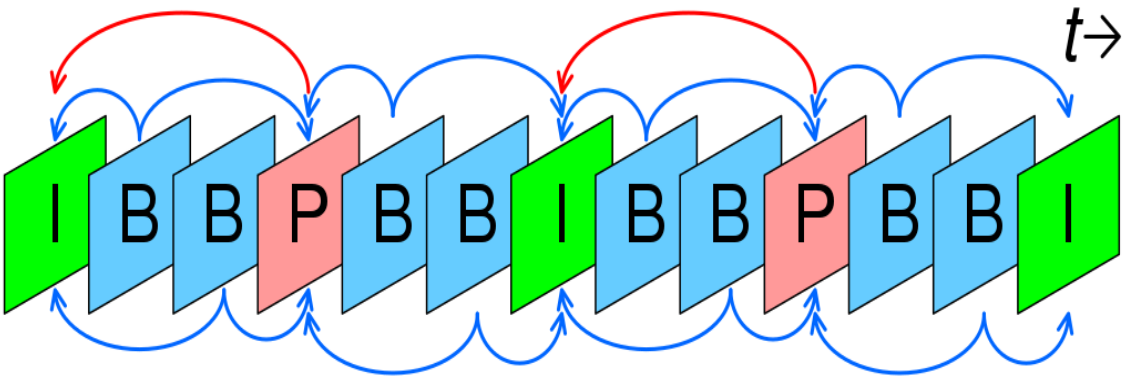

Figura 5: Exemplo do posicionamento e relação entre os quadros I, B e P na *timeline* de um vídeo. https://goo.gl/VAVPG2. Acessado em: novembro de 2018.

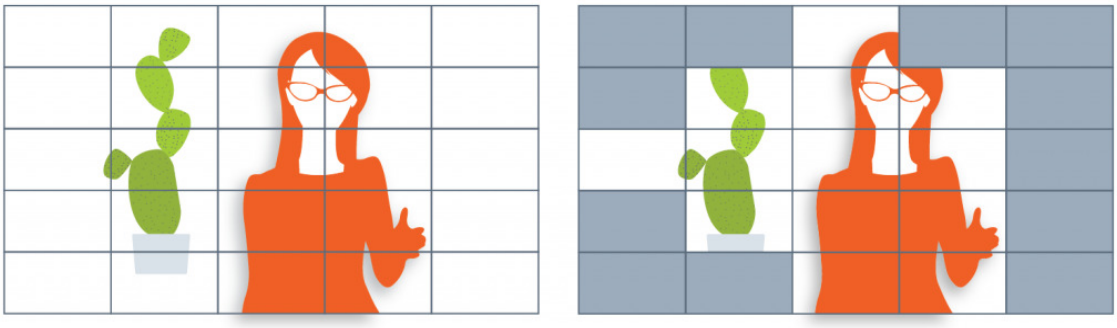

Figura 6: Exemplo da mudança de informação entre dois quadros. A imagem da esquerda representa o quadro I, a da direita representa o quadro B, os retângulos cinzas indicam áreas que devem ser extraídas do quadro I e posicionadas no B, pois não sofreram alteração. https://goo.gl/igPhs2. Acessado em: novembro de 2018.

2.1.3 Exibição dos frames

Após a extração do conteúdo comprimido, a informação deve ser exibida ao usuário. Nesse ponto envia-se o conteúdo do meio digital para o analógico. Cada *frame* é extraído na forma de uma matriz de pixels e que podem ser representados de múltiplas diferentes formas. O *pixel format*, como é denominado, pode ser representado como canais RGB, com a quantidade de *bits* variando entre 8 e 24 por canal. A Figura 7 ilustra uma visão separada de cada faixa. Outro padrão muito usado é o que utiliza os canais de luminância e crominância separadamente. Entre estes pode-se citar o YUV420, que utiliza o canal Y e que representa tonalidades de cinza, variando entre branco e preto e os canais U e V que representam as cores. A Figura 8 ilustra como esses canais são armazenados. Existem variações desse padrão que utilizam os canais U e V entrelaçados. Um exemplo desse é o NV12. Dessa forma, é possível formar a imagem e enviar a mesma ao gerenciador de vídeo do computador para que o quadro seja exibido na tela.

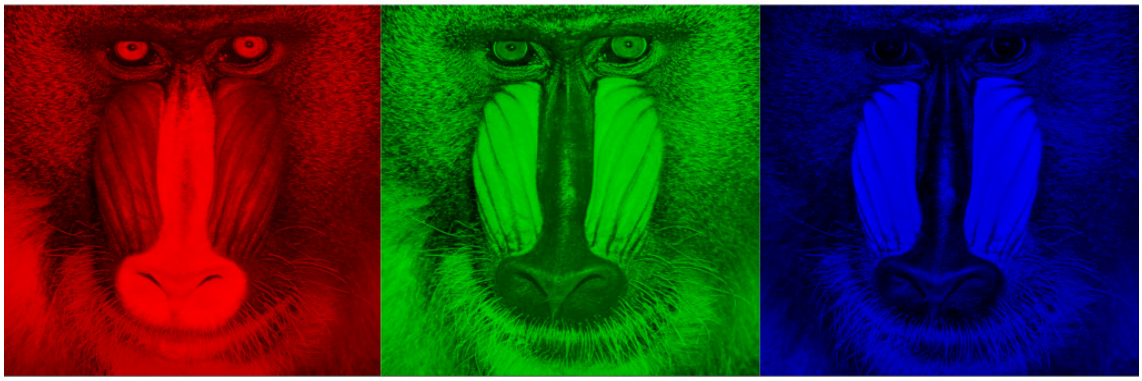

Figura 7: Canais R, G e B extraídos de uma imagem colorida. https://goo.gl/2dujqv. Acessado em: novembro de 2018.

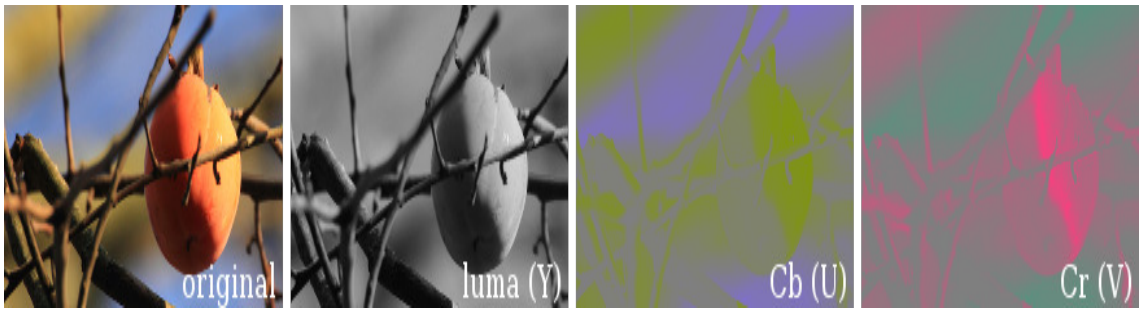

Figura 8: Canais Y, U e V extraídos de uma imagem colorida. https://goo.gl/7mcR4B. Acessado em: novembro de 2018.

2.1.4 *QR Code*

Quick Response code é um código de barras bidimensional criado no Japão no ano de 1994 que em seguida foi reconhecido pelos padrões AIM[2], JIS[3] e ISO[4][6][7]. O *QR code* foi criado com o intuito de manter o controle na manufatura de peças automotivas. Entretanto, o padrão ganhou popularidade pelo suporte a uma quantidade considerável de informação, por ser um padrão sem *royalties* e pela fácil detecção a partir de imagens. Esse tipo de código de barras 2D traz vários benefícios, dentre eles podemos citar:

- Suporte a captura 360º: O *QR code* é dotado de padrões de localização em três dos vértices do quadrado (ilustrado na Figura 9). Esse padrão permite que se detecte o ângulo de rotação, facilitando o reconhecimento do padrão gravado;

[2]https://www.aimglobal.org
[3]http://www.jisc.go.jp
[4]https://www.iso.org

- Resistência à distorção: O código pode ser escaneado independente do ângulo de visão e de curvaturas na superfície, o padrões de localização são distribuídos sobre o QR code à medida que a informação aumenta com o intuito de prover uma maneira para correção da distorção.

- Correção de erros: O algoritmo de Reed-Solomon é utilizado para interpretação de trechos com falha e o código pode ser lido mesmo se estiver borrado ou com uma seção em falta. Existem quatro níveis de correção variando entre sete e trinta por cento por área de símbolo.

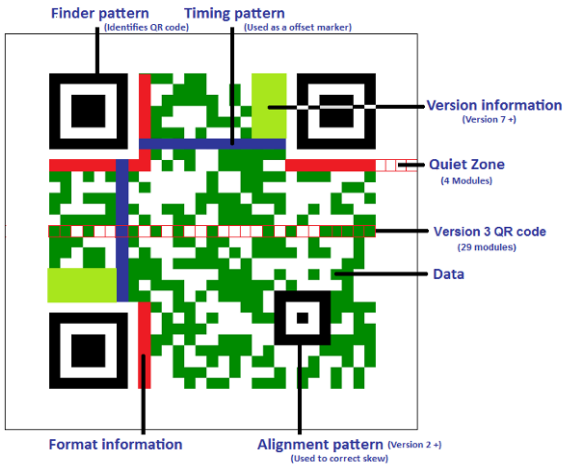

Figura 9: Exemplificação dos padrões do QR code. https://goo.gl/rEe95u. Acessado em: novembro de 2018.

3 TRABALHOS RELACIONADOS

Nesta seção serão apresentados trabalhos relacionados que são o estado da arte na exibição e manipulação de mídias distribuídas. Os trabalhos oscilam desde *players* distribuídos entre diversas máquinas a soluções que utilizam *browsers* para criar um ambiente multi aplicação.

Em seu trabalho, Kitamura[8] propõe uma solução que permite a transmissão de vídeos UHD de forma distribuída. Sua pesquisa mostra que a estratégia de sincronização utilizando essa abordagem é viável para mídias de grandes proporções e em seus testes foi possível a reprodução de vídeos variando do 4K ao 12K. Kitamura propõe o uso do codec JPEG2000 para comprimir os frames individualmente e utiliza uma abordagem master-slave para sincronizar os fragmentos do conteúdo com base na contagem dos frames.

Lucenildo[9] propõe uma abordagem que permite a manipulação de vídeos 4K. Segundo o autor, a solução é mais flexível, pois sua implementação adapta-se a diferentes *codecs* como o H.264 e o HEVC. Outro ponto importante é que o autor utiliza estereoscopia em sua proposta. Lucenildo[9] divide o vídeo em quatro seções que são transmitidas a um computador que decodifica esses fluxos, une os fragmentos de um quadro e os exibe em um projetor 4K.

Uma outra forma de manipular dados em um cenário multi tela, fundamenta-se na utilização de um *middleware* como o SAGE[10], o qual utiliza *browsers* em uma mesma máquina ou em máquinas distintas para criar um painel de vídeo de alta resolução. O propósito do SAGE é fornecer um ambiente onde múltiplas aplicações podem ser exibidas, funcionando como uma área de trabalho escalável. Deste modo, o foco da solução não é primariamente o suporte a vídeos UHD. Em seu trabalho, Lourenço[11] estuda a sincronização de vídeo 4K em múltiplas telas do SAGE2 e propõe uma solução capaz de sincronizar de forma mais eficaz a reprodução do conteúdo de vídeo. Entretanto, o SAGE é limitado a forma como o *browser* realiza a manipulação do vídeo.

Os trabalhos relacionados comprovam que o uso da divisão e conquista é viável para a reprodução em sincronia de vídeos UHD. Embora os resultados dos trabalhos sejam positivos, nenhuma das soluções permite a flexibilidade durante o playback. Destaca-se também a limitação no uso de apenas um *codec*, a uma única resolução ou a limitações do *middleware*.

4 SOLUÇÃO PROPOSTA

Para este trabalho foi desenvolvido um *player* de vídeo capaz de reproduzir vídeos UHD com suporte à aceleração de *hardware*. Para alcançar resoluções de nK o *player* deve ser executado de forma distribuída em diversos dispositivos e telas, sendo controlado por comandos de ajuste para que a partir de um observador a sincronia possa ser realizada.

O *player* apresenta robustez porque utiliza os codecs e containers mais populares no mercado. Todavia, apenas H.264, HEVC, VP9 e VP8 são atualmente compatíveis com a aceleração de *hardware*, que é imprescindível para a reprodução de vídeos com resolução superior a 4K. Quando habilitada, a aceleração utiliza instruções especializadas do processador gráfico, evitando o uso de instruções genéricas da decodificação em software. Além disso, quando a decodificação é feita diretamente pela GPU, os quadros que precisam ser decodificados são enviados diretamente às *surfaces* da placa que as decodificam utilizando os componentes físicos do computador.

Os benefícios da decodificação em hardware são facilmente percebidos ao reproduzir conteúdos UHD. Testes efetuados com vídeos 4K decodificados em software, que deveriam ter a taxa de exibição a 24 fps, passam a ter uma taxa de aproximadamente 4 fps. Se a aceleração de *hardware* é habilitada, pode-se exibir vídeos dessa dimensão em taxas superiores aos 24 fps sugeridos.

A razão de se utilizar uma solução distribuída é trazer escalabilidade ao sistema e, para isso, utiliza-se um mosaico de telas, que é formado por uma quantidade n de computadores, onde cada um deles é responsável por exibir um fragmento da imagem. Para manter a proporção correta, um número múltiplo de quatro é o ideal, uma vez que as resoluções de vídeo avançam nesse passo desde o *Full HD (High Definition)* como mostra a Figura 10.

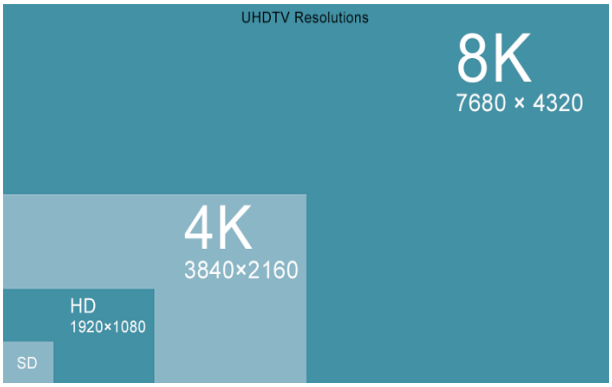

Figura 10: Comparação entre diferentes tipos de resolução de vídeo. https://goo.gl/7B8Nb2. Acessado em: Novembro de 2018

Na seção 4.1 a arquitetura do sistema será apresentada. Na seção 4.2 o player e seu funcionamento será detalhado e na seção 4.3 as características acerca do observador

da solução e sua relação com o *player* serão abordados.

4.1 ARQUITETURA

Além da arquitetura distribuída, baseamo-nos no modelo de comunicação *master-slave* para realizar a sincronia entre as partes. A primeira tela, situada no canto superior esquerdo, é considerada o *master*. As demais, chamadas de *slaves*, devem ajustar-se para exibir os *frames* conjuntamente com o quadrante de referência. Na Figura 11 é apresentada a organização das telas, chamadas de quadrantes, para formar o mosaico.

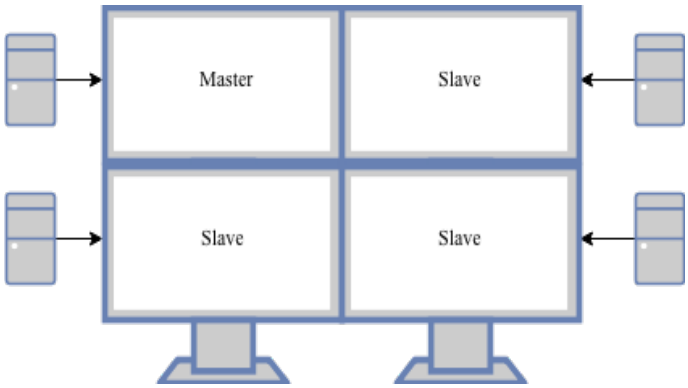

Figura 11: Representação da designação da função de cada computador que realiza a exibição.

Para realizar a exibição, o *player* de vídeo que foi desenvolvido permite a seleção de um conteúdo de qualquer resolução e *codec*. Entretanto, não são todos os vídeos que são suportados pela GPU que fará a decodificação. Deste modo, a mídia pode ser enviada para um decodificador em *software* ou para um que utilize o *hardware*.

Outra estratégia utilizada na solução proposta foi a apresentação da informação acerca do *frame* que está sendo exibido no *display* através do uso do *QR code*, cujos benefícios foram citados na seção 2.2. Quando requisitado o sistema habilita a exibição dos códigos como ilustrado na Figura 12.

O ajuste da sincronia é efetuado por um observador composto de uma câmera conectada a um computador que não faz parte do *cluster* de reprodução. A câmera captura quadros a uma taxa de 60 fps e os envia individualmente para o *software* que analisa a imagem buscando *QR codes*. Ao encontrar uma quantidade válida de dados o algoritmo envia os sinais de ajuste aos *slaves*.

4.2 *PLAYER*

Nesta seção serão descritos os detalhes do *player*, discutiremos todo o *pipeline* desde a demultiplexação da mídia até a exibição dos quadros na tela e a sua comunicação

Figura 12: Exemplificação de como os QR codes são exibidos na telas dos players.

com o observador.

Conforme explanado na seção 4, o *player* é flexível e pode trabalhar com um número extenso de *codecs*. Entretanto, optou-se pelo uso do *codec* H.264, escolha baseada no uso extensivo no mercado e pela sua maturidade.

O *player* quando inicializado deve consumir a mídia a partir de um arquivo ou através da rede. No segundo caso, os protocolos para *streaming* de vídeo UDP, TCP, HTTP e RTP são os mais populares.

No trabalho proposto a transmissão da mídia foi abstraída e houve um foco maior no ajuste da exibição a partir de um arquivo local. Deste modo, o primeiro passo caracteriza-se como a demultiplexação da mídia, que consiste na separação dos sinais que são transmitidos unidos, em múltiplas trilhas distintas. É importante salientar que um vídeo pode conter n trilhas, em seu container podem existir um ou mais vídeos, nenhuma, uma ou mais de uma trilha de áudio e de legenda.

Na implementação proposta o *container* é analisado pelo *player*, o qual extrai a trilha de vídeo e a envia para o processamento, conforme ilustrado na Figura 14.

Figura 13: Extração e envio da trilha de vídeo ao decodificador.

Antes de inicializar o processo de decodificação é necessário validar se o *codec* do vídeo que será enviado para *playback* é suportado pela GPU, que deverá fazer a decodificação com aceleração de *hardware*. Um processo exploratório é efetuado na mídia e

diversas informações são adquiridas, dentre elas estão o *frame rate* e o *codec* do conteúdo. A partir do *codec*, uma validação é realizada, através de uma API, a fim de validar se o processo poderá ser acelerado, pois havendo necessidade é possível realizar a decodificação em *software* do vídeo. Diversas APIs podem ser utilizadas para a habilitação de decodificação pelo meio físico, dentre elas pode-se destacar a VDPAU[5], VAAPI[6] e a Intel *Quick Sync Video*[7].

Destaca-se entre elas a VDPAU, uma vez que esta usa a placa de vídeo *offboard*, que são mais vantajosas do que as *onboard* pois possuem maior poder computacional e memória dedicada.

Como o trabalho proposto almeja uma solução de custo reduzido, optou-se por utilizar uma das demais bibliotecas, uma vez que estas são dedicadas a placas de vídeo *onboard*, o que implica em baixo custo e redução da complexidade da configuração do sistema operacional.

Em trabalhos anteriores o autor pôde utilizar a biblioteca provida pela Intel. No entanto, o esforço exigido para habilitar a sua funcionalidade em ambientes Linux é muito elevado, pois necessita da recompilação do *kernel* do sistema operacional para uma release específica e da instalação e manipulação de partes do sistema, o que não é intuitivo ao usuário leigo.

A opção selecionada para este trabalho é a VAAPI, uma biblioteca de código livre para GPUs *onboard* Intel. Ela apresenta excelente desempenho com mídias 4K e suporta os *codecs* mais populares do mercado, como VP9, H.264, HEVC e MPEG2. É importante destacar que o suporte a esses *codecs* depende da geração do processador, sendo os mais modernos capazes de trabalhar com os *codecs* informados anteriormente.

A passagem de fragmentos de mídia ainda comprimidos, assim como a de quadros prontos para a exibição é controlada por uma fila circular. A produção do conteúdo é intermitente, ou seja, quando a produção atinge 100% de ocupação da fila, ela entra em espera até que a capacidade decaia a um limiar especificado, que na solução proposta foi definido como metade do tamanho da fila. A Figura 14 exemplifica o funcionamento das filas.

[5]https://www.freedesktop.org/wiki/Software/VDPAU/

[6]https://01.org/vaapi

[7]https://www.intel.com.br/content/www/br/pt/architecture-and-technology/quick-sync-video/quick-sync-video-general.html

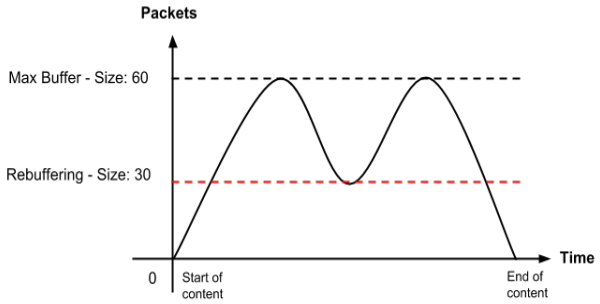

Figura 14: Exemplificação do funcionamento da fila circular.

Um pacote ainda comprimido é extraído da fila e enviado na ordem correta de decodificação à GPU. Essa ordenação faz uso do DTS, cujo funcionamento foi explicado na seção 2.

Quando um *frame* é corretamente decodificado pela GPU um *callback* é executado para que se possa ter acesso ao espaço de memória onde a matriz de píxeis que formam o quadro reside. A partir deste ponto, é possível o envio do quadro decodificado para a nova fila circular, o conteúdo é então processado e exibido de maneira apropriada.

A VAAPI utiliza por padrão o formato de pixel NV12, que foi descrito brevemente na seção 2.1.3. Para exibir o *frame* com essas características ao usuário utilizou-se a biblioteca SDL2, que consiste em um conjunto de ferramentas de desenvolvimento multi-plataforma desenvolvida para acesso a funcionalidades de áudio, periféricos como *joysticks* e gráficos através do uso do OpenGL[8] (Open Graphics Library) na maioria dos sistemas operacionais e do Direct3D[9] em ambientes Windows. Para que os frames possam ser exibidos, texturas do SDL são preenchidas com o frame recém decodificado e são em seguida enviados a uma *surface*, que é uma região onde as texturas são sobrepostas. Ao inicializar a exibição, um *clock* de referência com granularidade de microssegundos é estabelecido, o funcionamento deste relógio é interconectado ao *clock* da máquina, possibilitando passos equivalentes.

Para cada quadro o PTS é convertido para um valor em segundos, dado pela multiplicação do *timestamp* de apresentação com o *time base*, uma unidade de medida que representa os *ticks* do relógio. O tempo de espera até que um novo quadro possa ser exibido é dado pela diferença entre o PTS atual e o PTS anterior, acrescido do último tempo de apresentação decrescido do tempo atual do relógio. O resultado do algoritmo é fornecido em milissegundos. Dessa forma, o *player* deve esperar a duração desse intervalo até que o quadro possa ser exibido.

A fim de possibilitar uma experiência mais prazerosa ao usuário final, também foi

[8]https://www.opengl.org/
[9]https://docs.microsoft.com/en-us/windows/desktop/direct3d

implementada a funcionalidade de preenchimento de bordas, onde os vídeos são esticados para simular a reprodução do conteúdo por baixo da moldura das telas. A Figura 15 ilustra o cenário "sem borda".

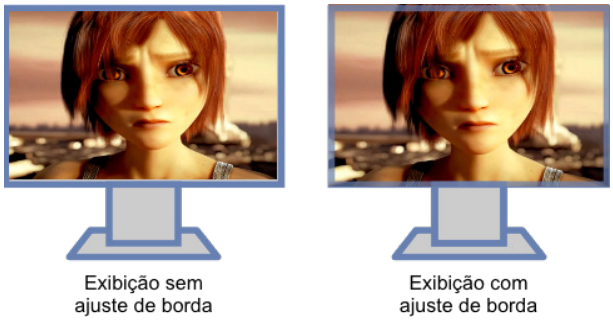

Figura 15: Representação da exibição do conteúdo com e sem o preenchimento de bordas

4.3 OBSERVADOR

Responsável por ajustar os quadrantes e mantê-los em sincronia, o observador é composto de um computador executando a funcionalidade de sincronia do player, uma placa de captura de vídeo e uma câmera filmadora. A câmera captura os quadros a uma taxa de 60 fps e resolução base de 860 x 480 pixels. Porém, quanto maior a resolução da captura, menor será a taxa de resultados que são extraídos. A biblioteca Zbar[10], que é utilizada para analisar os QR codes, deve varrer todos os pontos da imagem para buscar os padrões de localização dos códigos de barra. Além disso, ela deve aplicar os algoritmos de correção de erros, o que demanda um grande custo computacional quando escala-se a dimensão do *frame*. Para que somente *frames* mais novos sejam analisados, a fila circular implementada foi alterada com o intuito de sempre sobrescrever frames antigos se a produção for mais alta do que o consumo. Não é importante ter frames sequenciais para validar a discrepância entre *master* e *slaves*, uma vez que a diferença entre os relógios não mostra alterações em um curto intervalo de processamento.

A conexão entre observador e *players* é efetuada utilizando o protocolo TCP. Optou-se pelo seu uso pois ele garante a chegada dos dados ao cliente. Para o cenário proposto é essencial que os dados não sejam perdidos durante a transmissão. Se um quadrante não recebe a informação, o processo de sincronia demora ao menos mais um ciclo para fazer efeito.

Define-se por ciclo de processamento da solução, todo o *pipeline* desde a captura de n quadros, passando pela análise e extração da informação destes, até o envio do comando de sincronização e o ajuste dos *players*, independente destes estarem sincronizados ou não.

[10]http://zbar.sourceforge.net/

O endereço de IP dos players devem ser providos ao observador ordenados do quadrante 0 ao n, onde o quadrante 0, chamado de *master*, é situado no canto superior esquerdo. A contagem dos demais deve ser incremental e seguir para a direita até que uma linha acabe. Esse processo de contagem é retomada na próxima linha. A Figura 16 exemplifica o processo de numeração dos quadrantes.

A escolha dessa ordem é apenas uma convenção utilizada pelo autor e a numeração dos quadrantes é uma escolha do usuário, o que não altera o funcionamento da solução.

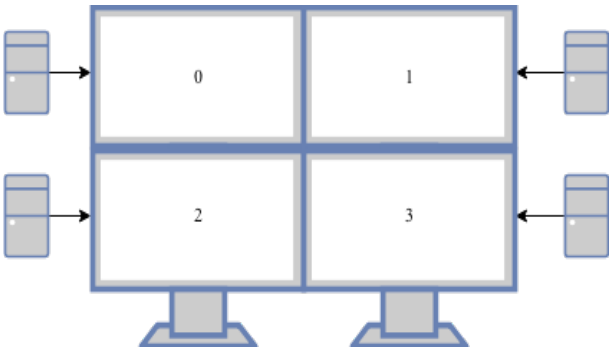

Figura 16: Exemplificação da numeração das telas.

Os quadros são capturados utilizando o OpenCV[11] (Open Source Computer Vision Library), que os converte para uma matriz monocromática em tons de cinza, requisito da biblioteca Zbar para que o processo de busca seja feito de forma mais eficiente. Ela então irá procurar os n códigos presentes na imagem e retornará valores indicando a sua posição através de duas coordenadas bidimensionais e também o conteúdo presente no corpo do código. Como os resultados são retornados fora de ordem, definiu-se um padrão para indicar qual informação pertence a cada quadrante. A máscara utilizada é a seguinte: "q=N:frame=M", onde N é o número do quadrante de onde esse *QR code* foi exibido e M é o *frame* que está sendo reproduzido.

Após receber os resultados referentes a um *frame*, deve-se validar se a quantidade de *QR codes* escaneados assemelha-se à quantidade de conexões efetuadas. Se os valores forem diferentes, tem-se mais conexões abertas do que *QR codes* encontrados. Logo, essa informação é considerada corrompida e o novo quadro deverá ser analisado. Se a quantidade de códigos for igual a de conexões, tem-se um grupo de informações válidas que são adicionadas ao conjunto de diferença de quadros. A partir de uma quantidade de informações no conjunto de diferenças pode-se estimar a variação entre o *master* e os demais quadrantes. O cálculo para definir a divergência de quadros é uma simples subtração entre um *master* e um *slave*. Se a subtração resultar em um valor positivo, deve-se comandar o *player* para descartar esse número de quadros para que a exibição acelere. Se o resultado for negativo, deve-se comandar ao *player* para manter um quadro

[11]https://opencv.org/

parado até que o tempo referente a quantidade de quadros desconformes seja atingida. Se o resultado for zero, os quadros estão em sincronia.

A comunicação entre observador e *player* é feita através de trocas de mensagens que indicam se os *QR codes* devem ser escondidos ou exibidos e se o *player* deve atrasar ou adiantar a sua reprodução. As mensagens possuem cinco *bytes*, onde o primeiro *byte* representa o comando que o *player* deve executar e os demais a diferença de *frames* com relação ao quadrante de referência (*Payload*). A Tabela 2 mostra os comandos aceitos pelo player.

Ação	**Comando**	***Payload***
Play/Pause	0x08	Vazio
Exibir *QR code*	0x01	Vazio
Esconder *QR code*	0x02	Vazio
Ajustar a exibição	0x04	Inteiro representando a discrepância com o*master*

Tabela 2: Ações e comandos suportados pelo *player*

Por fim, temos a arquitetura da solução proposta exemplificada na Figura 17, onde o observador captura os quadros do *player*, realiza a análise da imagem buscando *QR codes* e envia essa informação aos *players* para que o ajuste possa ser efetuado.

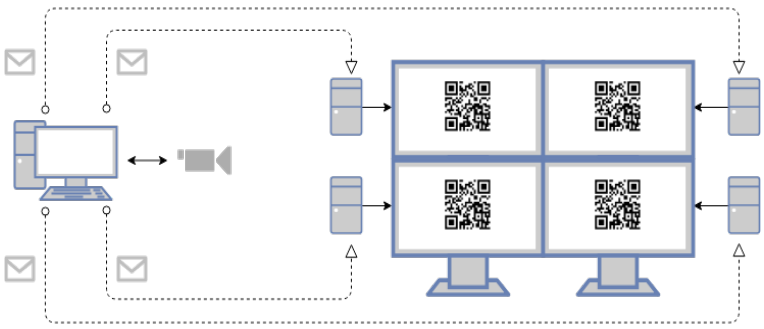

Figura 17: Arquitetura da solução proposta englobando observador e players.

5 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

No decorrer desta seção serão apresentadas as formas de validação da solução, assim como discussões sobre os resultados obtidos. Para a avaliação do funcionamento do sistema, três curtas foram utilizados com a finalidade de verificar a resiliência a resoluções e *frame rates* distintos. Conforme a Tabela 3, optou-se por utilizar conteúdos que partem do 4K com o fps variando entre 30 e 60, até um conteúdo 8K, 30 fps, disponibilizado em parceria pelo Professor Brett Stalbaum do departamento de Artes Visuais da Universidade da Califórnia em San Diego.

Vídeo	**Dimensão (Píxeis)**	**Duração**	***Frame rate***
A	3840 x 2160 (4K)	05m:38s	30
B	3840 x 2160 (4K)	05m:14s	60
C	7680 x 4320 (8K)	18m:50s	30

Tabela 3: Informação dos vídeos utilizados para a validação da solução

Nos testes foram utilizadas três máquinas Intel Nuc e uma máquina Dell OptiPlex para formar o *cluster* de exibição, as especificações técnicas podem ser encontradas na Tabela 4.

Máquina	**Processador**	**GPU**	**Memória**	**Armazenamento**	**Quadrantes**
Dell Optiplex	Intel Core i7 6700	Intel HD Graphics 530	8 Gigabytes	1 Terabyte	0
Intel NUC	Intel Core i5 5250u	Intel HD Graphics 6000	8 Gigabytes	120 Gigabytes	1, 2 e 3

Tabela 4: Configuração das máquinas utilizadas para executar os players

Utilizou-se como sistema operacional o Ubuntu 18.04 LTS para os *players*, testes foram efetuados com o Ubuntu 16.04 LTS, porém, a versão mais nova do sistema operacional apresenta maior fluidez na exibição do vídeo, tanto com o *software* implementado quanto com players de referência do mercado como o VLC.

O hardware utilizado pelo observador é composto de uma câmera filmadora Panasonic Lumix DMC-GH4, conectada a uma placa de captura de vídeo USB (Universal Seria Bus) INOGENI 4K2USB3. Para analisar os quadros capturados pela câmera, um notebook Asus x550LN foi utilizado para a execução do software do observador. A Tabela 5 apresenta as especificações técnicas desta máquina.

Máquina	**Processador**	**GPU**	**Memória**	**Armazenamento**
Asus x550LN	Intel core i5 4210u	Haswell-ULT	10 Gigabytes	240 Gigabytes

Tabela 5: Configuração da máquinas utilizada para executar o observador

Para realizar a exibição do conteúdo utilizou-se um *video wall* composto de quatro monitores. A princípio, as quatro telas iniciam de forma assíncrona, conforme a Figura 18.

Seguindo a mesma numeração exposta na seção 4.3, o quadrante número um é iniciado a frente do *master* e os quadrantes dois e três são iniciados após. A validação da sincronia foi efetuada de três formas distintas, que são:

- Observador: Funcionalidade do *software* que verifica a discrepância entre os quadros com base nas informações dos *QR codes*;
- Gravação: Consiste em capturar a exibição do conteúdo com câmeras de alto frame rate para analisar a informação textual presente na tela e validar o observador;
- Usuários: A opinião de um usuário é importante para definir se a apresentação do conteúdo ocorre de forma fluida.

Figura 18: Telas dessincronizadas. A tela 1 apresenta quadros a frente da referência e as telas 2 e 3 apresentam quadros prévios a referência.

Nos primeiros testes efetuados, optou-se por enviar o comando de ajuste para cada quadro capturado que fosse considerado válido, ou seja, se o algoritmo analisar o frame e encontrar os *n QR codes* solicitados, inicia-se a sincronia.

Entretanto, esse tipo de estratégia mostrou-se pouco eficiente, uma vez que dentre as muitas capturas que afirmavam que os quadros estavam sincronizados residiam informações que indicavam que existia a diferença de um quadro dentre o *master* e algum dos *slaves*. Com a ajuda de uma gravação e de usuários constatou-se que os quadrantes estavam sim, em sua maioria, em sincronia. Logo, supôs-se que o problema se dava pela diferença entre a taxa de atualização do monitor e a captura da câmera.

Existem casos em que a câmera captura um momento de transição da imagem, ou seja, os quadros estão sendo exibidos com alta precisão. Porém, durante a mudança de quadros, onde pode haver diferenças sutis (inferiores a 1/fps), há a captura de um *frame* pela câmera, portanto, os quadrantes estão em relativa sincronia. A Figura 19 exemplifica

o cenário em que os quadros são alterados do frame 10 para o 11. O *frame* adquirido pela câmera durante essa mudança indica uma discrepância ínfima, que de acordo com os testes com os usuários se mostrou imperceptível.

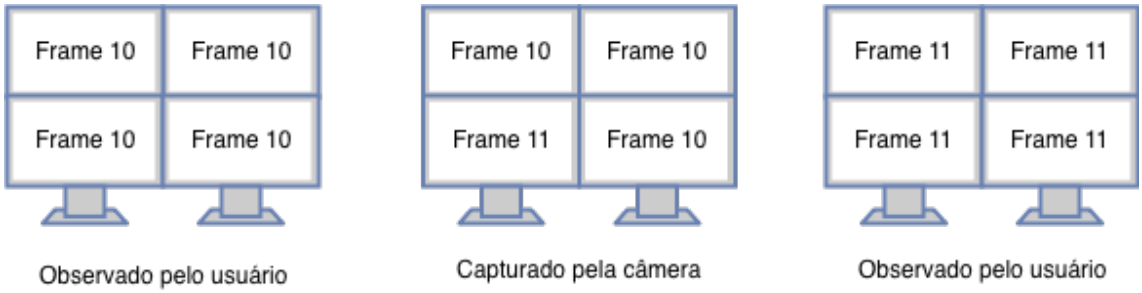

Figura 19: Exemplificação da captura da câmera durante a atualização do monitor.

Para contornar esse tipo de problema a estratégia foi alterada. Ao invés de enviar comandos de sincronia a cada quadro, decidiu-se calcular a moda de n capturas. Essa abordagem fez com que a quantidade de tentativas de sincronia reduzisse e solucionou o problema citado anteriormente.

A princípio, foi utilizado um total de 10 capturas para o cálculo da moda. Porém, ao observar o comportamento do player percebeu-se que os falso-positivos persistiram, logo o número de capturas foi escalado até o valor de 50 amostras, onde os erros de um quadro de diferença, deixaram de ser percebidos.

A fim de evitar o reenvio dos comandos durante o processo de sincronia, decidiu-se que os quadrantes devem apresentar uma tela preta durante o ajuste, evitando que os quadros sejam sincronizados novamente. Se esse cenário fosse ignorado, o processo de sincronia não chegaria ao fim, uma vez que o novo comando de ajuste iria sobrepor o anterior. Conforme ilustrado na Figura 19, o quadrante que está a frente do *master* exibe a informação do *frame* atual com o texto esverdeado, indicando que irá aguardar a diferença de tempo entre os quadros. Os quadrantes que estão atrasados apresentam um texto avermelhado, indicando que estão descartando quadros na tentativa de alcançar o *master*.

Figura 20: Telas durante o processo de sincronia. Quadrantes com o texto avermelhado descartam quadros, os com o texto esverdeados aguardam durante um intervalo de tempo.

Os testes realizados com as mídias A e C foram satisfatórios, pois após 3 ciclos de ajuste o conteúdo entre as telas encontra-se sincronizado. Entretanto, com a mídia B não foi possível alcançar os mesmos resultados. Ao tentar realizar o descarte de quadros para realizar a aceleração do conteúdo, verificou-se que a tela nunca alcançava o quadrante de referência. Logo, conclui-se que o decodificador acelerado trabalha de forma mais eficiente com mídias de maior resolução do que com conteúdos de alto *frame rate*. Outro problema causado por esse tipo de conteúdo diz respeito à captura das amostras. Quanto maior a taxa de reprodução do conteúdo, mais difícil é a análise dos *frames*. Conforme ilustrado na Figura 21, a câmera captura mais *frames* distorcidos do que *frames* bem formados. Dessa forma, o tempo até que seja possível enviar o comando de sincronia aos quadrantes é muito maior do que o das mídias de 30 fps.

Figura 21: QR codes capturados durante a atualização das telas.

Após o processo de ajuste, aguarda-se a validação do observador para que a mídia possa seguir seu fluxo sem alterações, a partir desse ponto os QR codes são escondidos.

A Figura 22 e a Figura 23 ilustram trechos distintos da exibição de um conteúdo após a remoção dos QR codes.

Figura 22: Telas sincronizadas com a solução de correção de bordas.

Figura 23: Telas sincronizadas com a solução de correção de bordas em um ponto futuro.

Para que a ideia seja mais palpável, um vídeo com a solução em execução pode ser encontrado em: https://youtu.be/WxWgX14UszY.

6 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho investigou-se uma solução para a reprodução e sincronização de mídias UHD. Assim, foi realizada uma implementação de um *player* escalável de ultra alta resolução e de um observador capaz de manter a exibição de fragmentos de um vídeo disposto em diferentes telas em concorrência.

Apesar de existirem problemas com mídias que possuem um alto *frame rate*, os testes realizados neste trabalho mostram-se satisfatórios e indicaram que o uso de um observador para realizar a sincronia das telas é uma estratégia promissora, pois possibilita a reprodução e sincronização de mídias UHD em aplicações do mundo real. No meio educacional e na medicina, área onde utiliza-se com maior peso aplicações relacionadas, uma adaptação da solução para a telemedicina, por exemplo, beneficiaria pacientes e estudantes.

Algumas alterações devem ser feitas no *player*, a transmissão do conteúdo através da rede é um passo importante que deve ser tomado para que a solução possa ocupar espaço de mercado. Também está no escopo de trabalhos futuros realizar testes com mídias de maior resolução, a possibilidade de migrar a solução para *hardwares* de custo ainda mais baixo como o Raspberry Pi[12], a inserção da capacidade de reproduzir áudio de forma distribuída e o desenvolvimento de uma versão para o sistema operacional Microsoft Windows.

[12]https://www.raspberrypi.org/

REFERÊNCIAS

[1] BECKER Valdecir; PACINE Bruno; LEMOS Guido. Evolução da Definição do Vídeo. *WebMedia*, page 4, 2014.

[2] WEYNAND Diana; PICCIN Vance; WEISE Marcus. *How Video Works: From Broadcast to the Cloud*. Focal Press, 3rd edition, 2015.

[3] TEKALP A. Murat. *Digital Video Processing*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2nd edition, 2015.

[4] READ Paul; MEYER Mark-Paul. *Restoration of Motion Picture Film*. Butterworth-Heinemann, 1st edition, 2015.

[5] HALSALL Fred. *Multimedia Communications*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 2000.

[6] LIU Y.; YANG J.; LIU M. Recognition of QR Code with Mobile Phones. *Chinese Control and Decision Conference (CCDC)*, pages 203–206, 2008.

[7] SOON T. QR Code. *Synthesis Journal*, pages 59–77, 2008.

[8] KITAMURA M.; SHIRAI D.; KANEKO K.; MUROOKA T.; SAWABE T.; FUJII T.; TAKAHARA A. Beyond 4K: 8K 60p live video streaming to multiple sites. *Future Generation Computer Systems 27*, pages 952–959, 2011.

[9] AQUINO Lucenildo; LEMOS Guido; GOMES Ruan; NETO Manoel; A. Duarte; R. Costa;. A Software-Based Solution for Distributing and Displaying 3D UHD Films. *IEEE*, pages 25–28, 2013.

[10] JEONG Byungil; RENAMBOT Luc; JAGODIC Ratko; SINGH Rajvikram; AGUILERA Julieta; JOHNSON Andrew; LEIGH Jason. High-performance dynamic graphics streaming for scalable adaptive graphics environment. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[11] LOURENÇO Claudio. Modelo de sincronização de vídeo para SAGE2 baseado em mídia adaptável. *Monografia apresentada a UFPB*, 2016.