Студент: Мустафин Владимир Михайлович

Группа : 17345

Научный руководитель: Таранцев Игорь Геннадьевич

Тематика работы:

«Разработка алгоритма поиска специфических сигнатур и их переходов в видеопотоке»

ОТЧЕТ

по проведению производственной практики, научно-исследовательской работы

Новосибирск, 2021

Оглавление

[Введение 3](#_Toc63600565)

[Обзор существующих решений 4](#_Toc63600566)

[1. Вычисление сигнатур 4](#_Toc63600567)

[1.1. Использование цветовой и яркостной гистограмм в качестве сигнатур 4](#_Toc63600568)

[1.2. Сигнатура на основе порядковой меры 5](#_Toc63600569)

[1.3. Сигнатура на основе использования быстрого преобразования Фурье 6](#_Toc63600570)

[Итоги 7](#_Toc63600571)

[2. Сравнение изображений 7](#_Toc63600572)

[2.1. Описание алгоритмов 8](#_Toc63600573)

[2.1.1. PSNR 8](#_Toc63600574)

[2.1.2. SSIM 9](#_Toc63600575)

[2.1.3. Сравнение сигнатур вычисленных с помощью быстрого преобразования Фурье 10](#_Toc63600576)

[2.2. Реализация и результаты. 10](#_Toc63600577)

[Итоги 15](#_Toc63600578)

[Архитектура алгоритма при добавлении ролика в базу данных 15](#_Toc63600579)

[1.1. Структура хранения части ролика: “VideoPart” 15](#_Toc63600580)

[1.2. Алгоритм сравнения кадра из потока видео с VideoPart 16](#_Toc63600581)

[Поиск пока VideoPart не найден: 16](#_Toc63600582)

[Поиск когда VideoPart уже найден: 16](#_Toc63600583)

[1.3. Алгоритм разделения ролика на части 17](#_Toc63600584)

[Архитектура решения 18](#_Toc63600585)

[Итоги 20](#_Toc63600586)

[Приложение 22](#_Toc63600587)

[Список литературы 24](#_Toc63600588)

# Введение

Текущая работа посвящена проблеме мониторингу информации, которая проходит по каналу прямого вещания. Такие каналы можно встретить как в интернете, так и на просторах интернета. Однако вопрос контроля показываемого изображения до сих пор остается актуальным.

Особенно актуальна это задача в сфере рекламного бизнеса. Контроль такого типа может ответить на различные вопросы рекламодателей, например: «Была ли показана моя реклама?», «Была ли показана моя реклама в полном объеме?», «В какое время была показана реклама?», и так далее. Данная работа посвящена разработки такого инструмента.

На рынке, вопросы такого типа не новы, и конечно же существуют альтернативные решения, однако большинство попыток автоматизировать данный процесс столкнулись с некоторыми проблемами, например:

* Скорость обработки данных,
* Стоимость оборудования,
* Стоимость отчетов,

и т.п. На данный момент, самый актуальной формой слежения является ручное подтверждение наличия рекламы, однако данное решения не пользуется большой популярностью из-за его стоимости.

Ключевым в данной работе является метод расчета сигнатур для каждого отдельного кадра. Сигнатура, это число с плавающей точкой, используемое для сравнения кадров. В зависимости от целей используются различные методы расчета сигнатур, каждый из методов обладает разными достатками и недостатками относительно друг друга. В текущей работе используется метод расчета сигнатур с помощью быстрого преобразования Фурье. Данный способ расчета сигнатур, показывает наиболее высокую устойчивость к шумам и достаточную, в рамках текущей задачи, уникальность для изображений.

Цель, которую в будущем станет возможным достичь с помощью данной работы, состоит в создании системы, позволяющей получать отчеты о присутствии того или иного видеоконтента в видеопотоке прямого вещания. Такая система должна иметь возможность применения для большого числа каналов одновременно и вмещать достаточное количество роликов для поиска.

Конкретной целью данной работы является оценка возможности разработки такой системы, при помощи алгоритма построения сигнатур. Как упоминалось ранее, идея не нова, однако предложенным методом обработки данных в данной работе, предстоит решить следующие проблемы:

* Обработка большого числа потоков одновременно,
* Достаточная скорость поиска контента в базе данных.

Для достижения поставленных целей необходимо решить несколько задач. Первая связана с уже существующим алгоритмом построения и сравнения сигнатур. Скорость работы предоставленного алгоритма не достаточна для решения поставленной задачи. Его необходимо доработать и оптимизировать. Данная оптимизация ожидается за счет изменения реализации алгоритма расчета сигнатур. Вторая задача состоит в разработке многопоточного алгоритма поиска и сравнения пар сигнатур (кадра из видео потока и кадра ролика из базы) в структуре хранения базы данных. В предложенном алгоритме будут использованы особенности сигнатур кадра, а именно:

* Слабое отличие сигнатур соседних кадров
* Сильное отличие при смене видеоряда.

Использование предложенных особенностей сильно снижает нагрузку на систему, а также ускоряет поиск. Для это требуется разработать алгоритм сравнения пар сигнатур (влево и право от интересующего кадра), а также разработать достаточно эффективный алгоритм поиска сигнатур в структуре базы данных.

На промежуточных этапах выполнения данной дипломной работы, есть необходимость создания приложений для тестирования скорости работы алгоритма, а также точности сравнения сигнатур. В итоге подразумевается разработка конечного, итогового решения, в котором будут показана точность и скорость обработки поступающий данных в реальном времени.

# Актуальность работы

В области сравнения изображений, основной проблемой является производительность. Предположим, у нас есть поток видео и база видеороликов, среди которых мы хотим найти совпадающие участки поступающего видеопотока. Характерные размеры:

* Кадр: пикселей(px) = 921600 px.
* Длинна ролика для поиска:
* Размеры видео базы: роликов кадров.
* CPU:  – операций в секунду.

## Попиксельное сравнение

Для попиксельного сравнения необходимо два такта процессора, вычитание и умножение (взятие модуля). Если отбросить остальные операции на попиксельное сравнение одного кадра необходимо 1843200 тактов. Путем несложных математический вычислений, становится понятно, что за секунду, данным алгоритмом возможно сравнить примерно 65 кадров. Так как в текущей работе предполагается база размеров кадров, данный алгоритм не подходит для поставленной задачи. Необходимо скорость увеличения скорости алгоритма в 76923 раза.

## Сравнение сигнатур

Алгоритм, предложенный в данной работе, выполняет сравнение сигнатур, за время . При такой скорости, появляется возможность сравнить примерно 200000 , что так же является недостаточным для нашей задачи. Необходимо скорость увеличения скорости алгоритма в 25 раза.

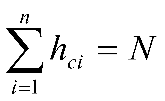
## Итоги

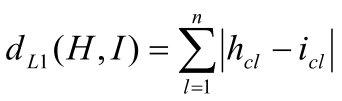
Из результатов, полученных выше, отчетливо понятно, что существующих алгоритмов сравнения кадров и их представлений недостаточно для решения поставленной задачи. Разработка такого алгоритма и является основной задачей данной работы.

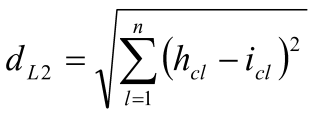
# Обзор существующих решений

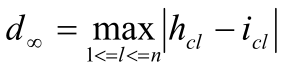
## Вычисление сигнатур

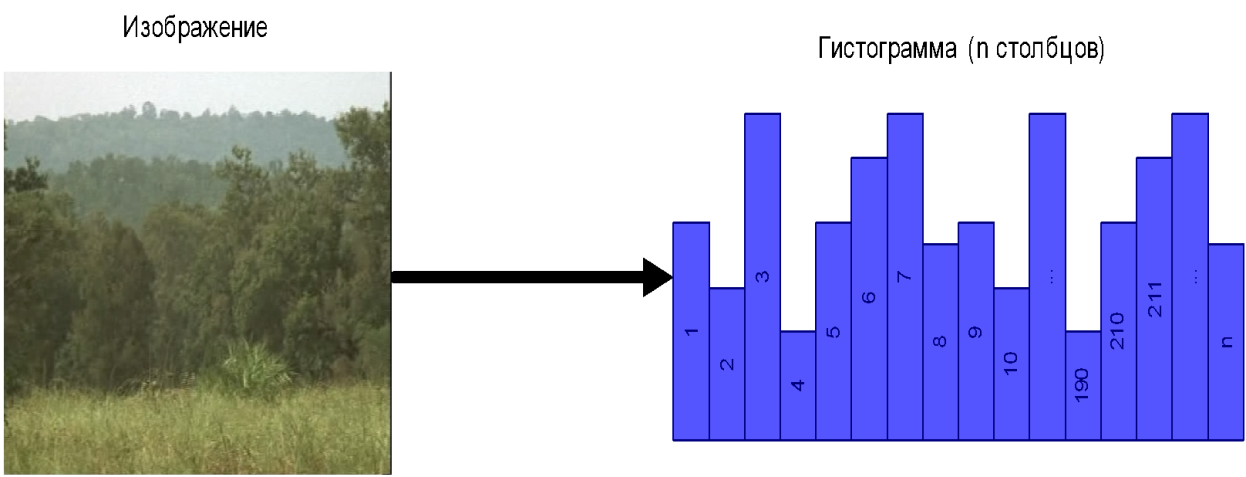
## Использование цветовой и яркостной гистограмм в качестве сигнатур

Использование яркостной и цветовой гистограмм, для обработки, а также подписи изображений является одним из самых старых и простых методов []. Так же, построенную яркостную гистограмму можно использовать в качестве сигнатуры изображения. Алгоритм состоит в том, чтобы отобразить цвета изображения *M* в дискретное цветовое пространство, содержащее *n* цветов, тогда цветовая гистограмма *H(M)* – это вектор ,где  является числом точек изображения *М* с цветом . Если *N* число точек на изображении *M*, тогда . Для определения расстояния между гистограммами *Н* и *I* используются различные функции:

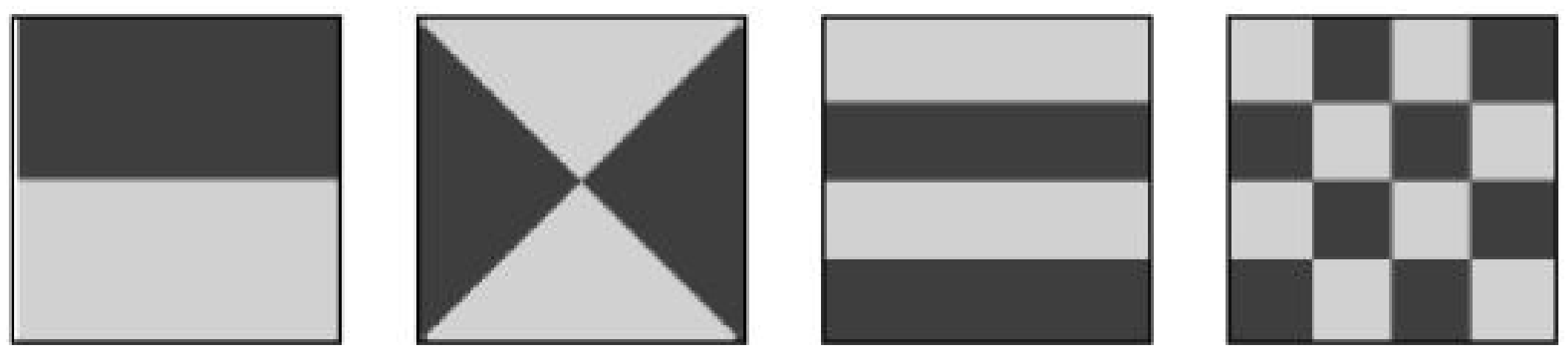








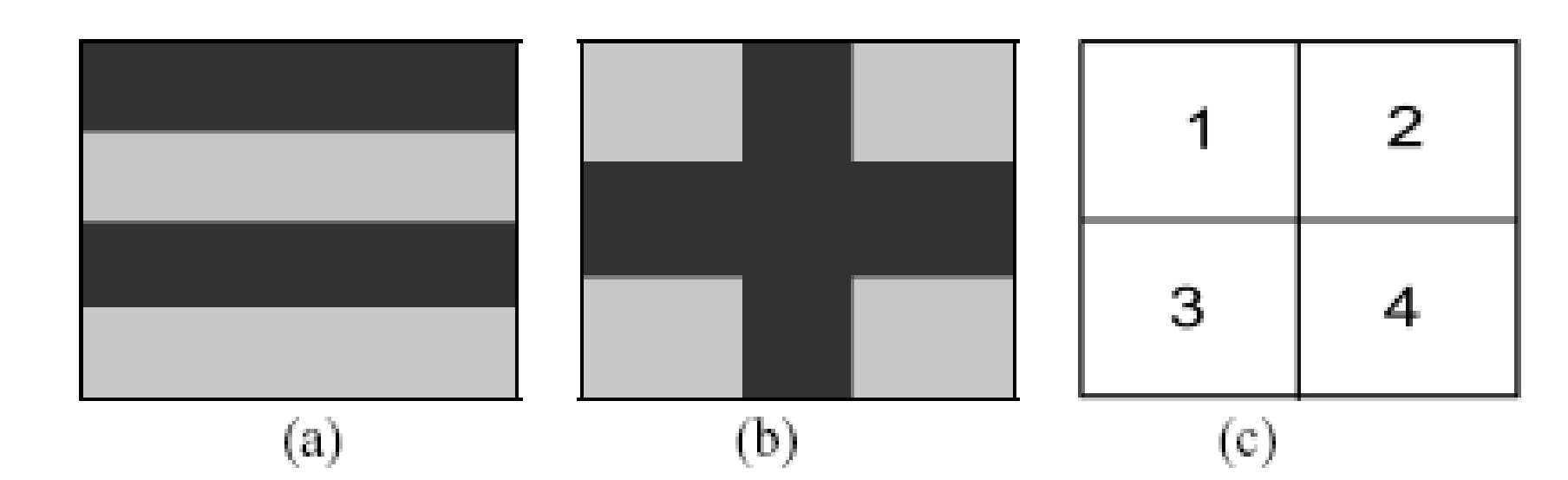
Цветовая гистограмма достаточно устойчива к шуму в изображении, а так же к поворотам и переносам. Но у неё есть несколько недостатков. Один из них, это то, что гистограмма описывает только распределение цвета или яркости на изображении, не учитывая при этом какую-либо пространственную информацию.



Также, если в кадре есть небольшой участок визуально важной информации, при использовании данного метода расчета сигнатур, данный участок может не внести существенного вклада в гистограмму и сравнение будет некорректным. Данный алгоритм в основном используется для разбиения последовательности изображения на похожие фрагменты, однако для нашей задачи он не походит.

## Сигнатура на основе порядковой меры

Алгоритм на основе порядковой меры [] является неким улучшение гистограммного метода. В данном алгоритме сначала изображение разбивается на N () областей. Для каждой из этих областей считается среднее значение яркости. Далее каждой области присваивается порядковый номер от нуля до N, в зависимости величины средней яркости. Вектор, компонентами которого являются эти числа, и есть сигнатура изображения. Основными преимуществами данной сигнатуры являются высокая скорость её вычисления, устойчивость к изменению яркости, а также некоторый учёт пространственной характеристики изображения. Однако существуют большое количество реальных изображений, которые имеют одинаковую порядковую меру, но совсем разное содержание. Это основная причина, по которой данный алгоритм не может быть использован в данной работе.



## Сигнатура на основе использования быстрого преобразования Фурье

Как дискретное преобразование Фурье (ДПФ), так и дискретно-косинусное преобразование (ДКП) широко используется при обработке видеосигнала. Известно, что большая часть информации об изображении содержится в низкочастотных коэффициентах ДПФ и ДКП. Поэтому эти преобразование позволяют уменьшить размерность при обработке изображения.

В алгоритме, рассмотренном в [1], используется трёхмерное ДКП для учёта пространственно-временных характеристик видео клипа. Сначала производится преобразование видео последовательности к единому пространственно-временному формату, в результате которого получается последовательность из 64 изображений, размерами 32х32 пикселя. Далее из них составляется матрица 32х32х64, над которой выполняется трёхмерное дискретно-косинусное преобразование. Для получения сигнатуры используются 64 низкочастотных коэффициентов ДКП (, i = 0..63) из левой верхней подматрицы размером 4х4х4. Далее производится сортировка и квантование коэффициентов. Все коэффициенты, которые больше медианы объявляются равными 1, меньшие же коэффициенты – 0. Медиана считается следующим образом , где  - отсортированный список низкочастотных коэффициентов ДКП. Одно битное квантование добавляет устойчивости к небольшим различиям в сигнатуре. Для определения дистанции между сигнатурами в рассмотренной работе используется расстояние Хэмминга — мера различия объектов одинаковой размерности (, где  – объекты размерности p). Результаты экспериментов, проведённых в рассмотренной работе, показывают, что полученная таким образом сигнатура является устойчивой ко многим типам шумов (размывание, изменение яркости, небольшие геометрические изменения, артефакты сжатия), а также обладает уникальностью, т.е. она различна у двух разных изображений.

В работе [2] используется подобный метод построения сигнатуры. Существуют несколько отличий. Во-первых, не учитываются временные характеристики, т.е. ДКП выполняется над двумерной матрицей, представляющей собой обрабатываемое изображение. Во-вторых, квантование происходит относительно среднего значения подматрицы низкочастотных коэффициентов ДКП, т.е. , где nxn – размерность подматрицы (в работе n = 5..8). В качестве функции сравнения, так же, как и в [1], использовалось расстояние Хэмминга.

При использовании данного алгоритма в выпускной работе на соискание степени магистра, выпускника кафедры автоматизации физико-технических исследований [3], в ходе работы, было рекомендовано использовать быстрое преобразование Фурье. В реализации алгоритма, используемого в текущей работе, в качестве сигнатуры были использованы коэффициенты низкочастотной подматрицы без выполнения их квантования. В этом случае в качестве функции сравнения в работе [3] было выбрано Эвклидово расстояние между двумя одномерными векторами размерности : .

## Итоги

Алгоритм использования цветовой и яркостной гистограмм в качестве сигнатур, не учитывает какую-либо пространственную информацию, а также небольшие участки визуально важной информации не вносят существенный вклад в сигнатуру, что может понести некорректное сравнение сигнатур, поэтому от данного алгоритма было решено отказаться.

Алгоритм расчёта сигнатур на основе пространственной меры, не обладает достаточной уникальностью, т.к. реальные изображения, могут иметь одинаковую пространственную меру, но совсем разное содержимое, поэтому от данного алгоритма было решено отказаться.

Из рассмотренных алгоритмов, наиболее устойчивым к многим типам шумов, таким как, размывание, изменение яркости, небольшие геометрические изменения, артефакты сжатия и т.п., является алгоритм вычисления сигнатур на основе быстрого преобразования Фурье. Так же, данный алгоритм обладает высоким показателем уникальности [1]. Данный алгоритм был выбран в качестве основного в текущей работе.

## Сравнение изображений

Одной из задач, в данной работе является сравнение сигнатур, вычисленным по кадрам. Далее будут рассмотрены методы прямого сравнения изображений, без вычисления сигнатур, а также сравнение времени работы с выбранным алгоритмом.

На просторах интернета, для сравнения изображений используют в основном две методики PSNR (peak signal-to-noise ratio), пиковое отношение сигнала к шуму и SSIM (structure similarity), индекс структурного сходства. Для проведения экспериментальных исследований было использовано два рекламных ролика, “Mercedes” и “S7”. В оригинале эти видео скачивались без каких-либо субтитров, надписей и так далее. В данном эксперименте основной целью было понять, насколько надписи, субтитры влияют на показатели данных алгоритмов и на алгоритм, предложенный в данной работе. Далее представлены кадры с вставленным фрагментами:



Кадр с вставленной надписью (BigTitle)



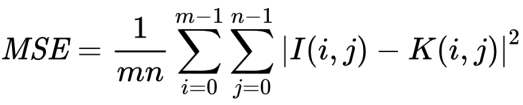
Кадр с вставленной бегущей строкой(Ticker)

## Описание алгоритмов

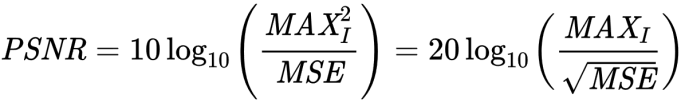
## PSNR

PSNR наиболее часто используется для измерения уровня искажений при сжатии изображений. Проще всего его определить через среднеквадратичную ошибку (СКО) или MSE (mean square error).

В случае использования MSE этот показатель для двух монохромных изображений *I* и *K* размера *m*×*n*, одно из которых считается зашумленным приближением другого, вычисляется по формуле:



PSNR определяется так:



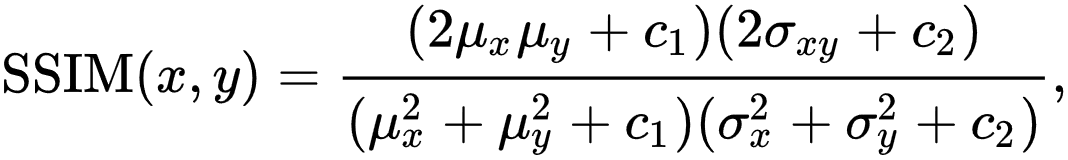
где *MAXI* — это максимальное значение, принимаемое пикселем изображения. Когда пиксели имеют разрядность 8 бит, MAXI = 255. Вообще говоря, когда значения сигнала представлены линейно (PCM) с B битами на значение, максимально возможное значение *MAXI* будет 2B-1. Для цветных изображений с тремя компонентами RGB на пиксель применяется такое же определение PSNR, но MSE считается по всем трем компонентам (и делится на утроенный размер изображения). Типичные значения PSNR для сжатия изображений лежат в пределах от 30 до 40 dB

## SSIM

Индекс структурного сходства (SSIM от англ. structure similarity) является одним из методов измерения схожести между двумя изображениями. SSIM-индекс это метод полного сопоставления, другими словами, он проводит измерение качества на основе исходного изображения (не сжатого или без искажений). SSIM-индекс является развитием традиционных методов, таких как PSNR (peak signal-to-noise ratio) и метод среднеквадратичной ошибки MSE, которые оказались несовместимы с физиологией человеческого восприятия.

Отличительной особенностью метода, помимо упомянутых ранее (MSE и PSNR), является то, что метод учитывает «восприятие ошибки» благодаря учёту структурного изменения информации. Идея заключается в том, что пиксели имеют сильную взаимосвязь, особенно когда они близки пространственно. Данные зависимости несут важную информацию о структуре объектов и о сцене в целом.

SSIM метрика рассчитана на различные размеры окна. Разница между двумя окнами *x* и *y* имеющими одинаковый размер *N×N*:



где:

* – среднее *х,*
* – среднее *y,*
* – дисперсия *x,*
* – дисперсия *y,*
* – ковариация *x* и *y,*
* – две переменных:
  + *–* динамический диапазон пикселей
  + и – константы.

Приведённая формула применима только для яркости изображения, по которой и происходит оценка качества. Полученный SSIM-индекс лежит в пределах от 0 до +1. Значение +1 достигается только при полной аутентичности образцов. Как правило, метрика рассчитана на окно размером 8×8 пикселей. Окно может смещаться через пиксель, но специалисты рекомендуют использовать группы окон для уменьшения сложности вычислений.

## Сравнение сигнатур вычисленных с помощью быстрого преобразования Фурье

В алгоритме, выбранном в качестве основного в данной работе, функция сравнения двух сигнатур была взята как, Эвклидово расстояние между двумя одномерными векторами размерности: : .

## Реализация и результаты.

Для реализации я использовал библиотеку для работы с видео и изображениями – OpenCv и вот что у меня получилось:

PSNR:

double getPSNR(const Mat& I1, const Mat& I2)

{

Mat s1;

absdiff(I1, I2, s1); // |I1 - I2|

s1.convertTo(s1, CV\_32F); // cannot make a square on 8 bits

s1 = s1.mul(s1); // |I1 - I2|^2

Scalar s = sum(s1); // sum elements per channel

double sse = s.val[0] + s.val[1] + s.val[2]; // sum channels

if (sse <= 1e-10) // for small values return zero

return 0;

else

{

double mse = sse / (double)(I1.channels() \* I1.total());

double psnr = 10.0 \* log10((255 \* 255) / mse);

return psnr;

}

}

MSSIM (M в начале это mean):

Scalar getMSSIM(const Mat& i1, const Mat& i2)

{

const double C1 = 6.5025, C2 = 58.5225;

int d = CV\_32F;

Mat I1, I2;

i1.convertTo(I1, d);

i2.convertTo(I2, d);

Mat I2\_2 = I2.mul(I2); // I2^2

Mat I1\_2 = I1.mul(I1); // I1^2

Mat I1\_I2 = I1.mul(I2); // I1 \* I2

GaussianBlur(I1, mu1, Size(11, 11), 1.5);

GaussianBlur(I2, mu2, Size(11, 11), 1.5);

Mat mu1\_2 = mu1.mul(mu1);

Mat mu2\_2 = mu2.mul(mu2);

Mat mu1\_mu2 = mu1.mul(mu2);

Mat sigma1\_2, sigma2\_2, sigma12;

GaussianBlur(I1\_2, sigma1\_2, Size(11, 11), 1.5);

sigma1\_2 -= mu1\_2;

GaussianBlur(I2\_2, sigma2\_2, Size(11, 11), 1.5);

sigma2\_2 -= mu2\_2;

GaussianBlur(I1\_I2, sigma12, Size(11, 11), 1.5);

sigma12 -= mu1\_mu2;

Mat t1, t2, t3;

t1 = 2 \* mu1\_mu2 + C1;

t2 = 2 \* sigma12 + C2;

t3 = t1.mul(t2); // t3 = ((2\*mu1\_mu2 + C1).\*(2\*sigma12 + C2))

t1 = mu1\_2 + mu2\_2 + C1;

t2 = sigma1\_2 + sigma2\_2 + C2;

t1 = t1.mul(t2); // t1 =((mu1\_2 + mu2\_2 + C1).\*(sigma1\_2 + sigma2\_2 + C2))

Mat ssim\_map;

divide(t3, t1, ssim\_map); // ssim\_map = t3./t1;

return mean(ssim\_map);

}

Далее с измененными роликами проводились ряд экспериментов, а именно сравнение скорости работы, а также визуальное сравнение качество работы, и вот результаты:

Как мы видим из графиков, среднее время сравнения сигнатур на основе алгоритма быстрого преобразования Фурье (Comp Sign), предложенного в данной работе: 5 мкс, что является приемлемым для поставленных целей. Так же стоит отметить, что при работе в Visual Studio на платформе Windows, под выполняемый процесс отдается лишь 30% ресурсов процессора, что конечно же сильно влияет на данные показатели. Сравнение изображений методом PSNR и SSIM выполнялось на MacOS и на более современном процессоре, однако при этом, время выполнения вычисления сигнатуры SSIM оказалось около 90000 мкс, это связано с вычислением , с помощью метода GaussianBlur. Вычисление методом PSNR оказалось самым приемлемым, и среднее время составило: 9000 мкс.

Теперь давайте взглянем на результаты, которые показали данные алгоритмы. Напомню, что сравнения происходило по двум роликам, один – оригинальный (без надписей, субтитров и т.д.), второй измененный.

Заголовок (BigTitle), появляется на видео на 7 – ой секунде, т.к. видео 30 кадр/с, изменения начинаются примерно на 210 кадре, что отлично видно из всех графиков. Алгоритм MSSIM обладает самым большим количеством информации об изменении в кадре, однако скорость вычисления не позволяет использовать его в нашей работе. PSNR изначально разрабатывался для определения качества сжатия изображения, что делает этот алгоритм неустойчивым к шумам. Так же алгоритм PSNR, не дает никакой структурной и пространственный информации, что делает его неустойчивым к таким изменениям в входящем изображении.

Графики остальных экспериментов находятся в приложении.

## Итоги

Алгоритм PSNR является неустойчивым к шумам и не дает никакой структурной информации изображения. Эксперименты с алгоритмом MSSIM, показали недопустимое для данной работы время выполнения операции. Поэтому от этих алгоритмов было решено отказаться.

Использование Эвклидова расстояния между двумя одномерными векторами в качестве функции сравнения сигнатур, показало, как и лучшее время сравнения, так и высокие показатели информативности. Именно этот метод в совокупности с алгоритмом быстрого преобразования Фурье для вычисления сигнатур, были выбраны в данной работе.

# Архитектура алгоритма при добавлении ролика в базу данных

Перед тем, как добавить новый рекламный ролик в базу данных, его необходимо подготовить. Это делается для ускорения поиска сигнатур по базе данных.

Основная идея, состоит в том, что любой ролик можно разбить на части. Части будут строиться по принципу схожести, т.е. если в ролике подряд идут похожие кадры, они будут добавляться в одну и ту же часть, если порог, при сравнении сигнатур, больше заданного, то будем считать, что началась следующая часть. Порог в текущем исследовании будет носить рекомендательный характер и может быть изменен пользователем.

Далее будут описаны структуры хранения частей ролика, а также алгоритм их наполнения.

Основные понятия и константы:

* VideoPart – структура хранения части ролика.

* – порог сравнения первого и последнего кадра из VideoPart.

* – порог сжатия. Данный порог необходимо вычислить эмпирически, путем сравнения исходного видео и после различных популярных алгоритмов сжатия.

* – порог наличия. Данный порог необходимо вычислить так же эмпирически. Порог будет использоваться для определения, находится ли искомый кадр в VideoPart. Предполагается , а также .
* mainSignature – сигнатура, являющаяся представителем ячейки VideoPart.

## Структура хранения части ролика: “VideoPart”

За хранение набора сигнатур, вычисленных из кадров подготавливаемого ролика, отвечает класс **SignatureHandler**. А именно в нем хранится динамический массив сигнатур. Так как поиск в массиве происходит за константное время O(1), в создаваемой структуре не будет хранится сигнатура, а лишь разметка. При этом у класса **SignatureHandler** будет константный метод **getSignature(size\_t index)**, который будет возвращать необходимую сигнатуру по индексу.

Далее будут описаны поля структуры хранения части ролика:

* Номер ключевого кадра – кадр подготавливаемого ролика, на котором произошел резкий скачок значения сравнения сигнатур соседних кадров (а именно разница оказалась выше задаваемого порога)(mainSignatureIndex).
* Номер последнего кадра, входящего в промежуток этой части(lastFrameIndex).
* Указатель на хранилище сигнатур, а именно класс **SignatureHandler**
* isMain – булева переменная. True, означает что данный videoPart построен на резком переходе видео. False, что текущий videoPart построен на основе ограничений , отличия первого и последнего кадров videoPart.

Описанного выше набора полей структуры достаточно для описания части ролика.

## Алгоритм сравнения кадра из потока видео с VideoPart

Поток видео: stream,

Текущий кадр stream: streamFrame,

Расстояние между streamFrame и mainSignature: mainDiff.

Поиск выполняется внутри структуры хранения базы роликов.

isVideoPartFound = false; - булевая переменная для объединения двух алгоритмов в StateMachine.

currentIndex – текущий индекс в структуре videoPart, что по сути тоже самое, что индекс в SignatureHandle, ведь в videoPart у нас только начало и конец.

Структура searchWindow, используется для поиска кадров в режиме “окно”. Для стандартных случаев, это +- 2 кадра (стандартная возможная ошибка пропуска кадров), в других случаях данный размер может быть произвольным.

Поля структуры searchWindow:

* leftLimit – количество кадров влево от текущего
* rightLimit – количество кадров вправо от текущего

While(!stream.isEmpty()){

Соответственно для начала работы StateMachine мы получаем кадр

streamFrame = stream.getNextFrame();

if(!isVideoPartFound){

### Поиск пока VideoPart не найден:

If(mainDiff < ){

Считаем, что этот кадр и есть искомый. Добавляем VideoPart в массив совпавших частей ролика, и помечаем mainSignatureIndex как начало совпавшего промежутка (beginIndex).

isVideoPartFound = true;

currentIndex = beginIndex;

}else if (mainDiff < ){

defineSearchWindow(currentIndex, videoPart); - псевдо реализация ниже.

checkFrameInWindow(); - описание ниже (сделал как ссылку, сразу перекинет)

Данное условие говорит о том, что кадры похожи, но недостаточно, чтобы быть уверенным, что это искомый кадр. Мы предполагаем, что кадр может находится внутри ячейки VideoPart. Для этого необходимо пройти по всему VideoPart. Если разница между сигнатурой streamFrame и сигнатурой кадра(frameFromCurrentVideoPart) из videoPart < , кадр найден. Добавляем VideoPart в массив совпавших частей ролика, и помечаем frameFromCurrentVideoPart, как начало совпавшего промежутка (beginIndex), а так же isVideoPartFound = true, currentIndex = beginIndex;

Иначе, кадра в этом VideoPart нет, продолжаем поиск, а также isVideoPartFound = false;

}

}else{

### Поиск когда VideoPart уже найден:

currentIndex += 1;

curSignature: сигнатура полученная по индексу currentIndex из VideoPart.

errorVector ; - вектор пропущенных кадров.

while(currentIndex <= lastFrameIndex){ - идем пока не дошли до конца VideoPart

diff = getDiffOfSignatures(streamFrame, curSignature);

if(diff < ){

считаем что данный кадр и есть искомый.

isVideoPartFound = true;

continue; (продолжаем сравнение)

}else{

trulyIndex = checkFramesInWindow(); - в это функции проверяются кадры вошедшие в окно, различие сигнатур с streamFrame окажется меньше , возвращаем его индекс, иначе текущий. Если таких несколько, возвращаем минимальный. Так же произвожится корректировка окна.

If(trulyIndex != currentIndex ){

errorVector.pushback(currentIndex);

currentIndex = trulyIndex;

}else{

Считаем что с текущим VideoPart работа закончена, возвращаемся к пункту: “Поиск пока VideoPart не найден”.

isVideoPartFound = false;

Заканчиваем работу, закрывая промежуток кадром currentIndex - 1;

}

}

}

if(currentIndex == lastFrameIndex){

Данный блок означает, что кадры из текущего VideoPart закончились, мы ожидаем следующий VideoPart из текущего SignatureHandler.

Массив структур VideoPart, так же будет храниться в SignatureHandler, поэтому пытаемся получить следующий. Если не получилось, закрываем промежуток:

currentIndex

isVideoPartFound = false;

Иначе продолжаем работу.

}

}

}

defineSearchWindow(currentIndex, videoPart){

if(videoPart.isMain()){

searchWindow.leftLimit = 2;

searchWindow.rightLimit = 2;

}else{

rightSearch();

leftSearch();

//данные функции выполняют проход по массиву сигнатур( из videoPart.SignatureHangler) вправо и влево, пока порог streamFrame.signature и кадра не больше или не дошли до конца videoPart.

Про концы videoPart:

Предположим, мы попали в mainFrame (videoPart.isMain() = False), тогда идти дальше конца текущего videoPart и начала предыдущего, смысла нет, так как они построены так, что между первым и последним кадром разница .

Если мы попали не в mainframe, оном поиска становится:

searchWindow.leftLimit = getDiff(videoPart.mainFrame.index(),currentIndex)

searchWindow.rightLimit = getDiff(videoPart.lastFrame.index(),currentIndex)

}

}

checkFrameInWindow(){

checkSignatures() – идем слево – направо по окну, ищем кадры у которых порог меньше , среди них выбираем минимум

{

если такой кадр найден:

Оставляем окно текущим.

Если этот кадр следующий, за currentIndex (мы его и ожидали), то обрезаем окно до +- 2;

}

возвращаем этот кадр, иначе currentIndex

}

## Алгоритм разделения ролика на части

Далее будет описан алгоритм построения частей ролика:

Порог принадлежности: threshold. Предполагается  threshold.

Структура для хранения части ролика: newVideoPart.

Текущий индекс в массиве сигнатур: currentIndex = 0.

Массив сигнатур: signatures.

Массив частей ролика: videoPartHandler.

endVideoPart = false; булевая переменная, означающая что работа с текущим VideoPart окончена.

while(currentIndex != signatures.size()){

diff = getDiffOfSignatures(signatures[currentIndex], signatures[currentIndex + 1]);

If(diff <= threshold){

if(getDiffOfSignatures(signatures[currentIndex], newVideoPart.mainFrame) >= ){

endVideoPart = true;

}

}else if(diff == 1){

Данный случай необходимо исследовать отдельно, т.к. в курсовой работе [], был отмечен один из недостатков данного алгоритма, а именно: если после абсолютного черного кадра, идет кадр с слегка увеличенной яркостью, данный алгоритм показывает максимальное значение разницы кадров, что не является истинной.

If(checkIfBlack(signatures[currentIndex], signatures[currentIndex + 1]){

//блок обработки особого случая

}

}else{

endVideoPart = true;

}

if(endVideoPart){

newVideoPart.lastFrame = currentIndex;

videPartHandler.pushback(newVideoPart);

newVideoPart = VideoPart(параметры инициализации) ;

endVideoPart = false;

}

currentIndex ++;

}

# Архитектура решения

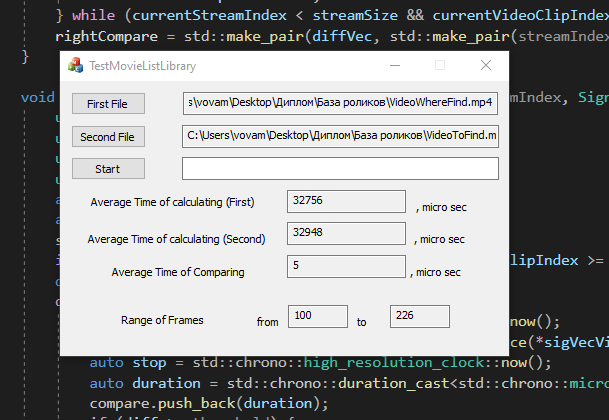
В данной главе я бы хотел описать свое видео задачи и как я его буду решать, а также что уже было сделано.

В итоговом решение планируется обработка большого количества видеопотоков одновременно, а именно расчета сигнатур для данного потока, После расчета сигнатур, будет выполнятся сравнение соседних кадров и при резком скачке значения разности сигнатур (а именно так обычно происходит переход к рекламным видеороликам), будет выполнятся поиск данного кадра (кадра справа от скачка) в базе данных. В реализации уже готовой программы есть класс **SignatureHandler** (<https://github.com/Geridok/Player/blob/main/TestMovieListLibrary/WorkWithFrames.h>), данный класс используется для хранения сигнатур, как сигнатур роликов для поиска, так и для сигнатур входящего потока (с определённым размером буфера, который необходимо будет определить). Данный класс добавляется в декодер видеопотока при инициализации. В структуре хранения базы данных планируется хранить индекс сигнатуры (которая отвечает за данную ячеку), а также указатель на этот класс, для передачи его уже в класс поиска совпадающего фрагмента видео.

Для поиска совпадающего фрамента, а именно сравнения пар сигнатур и сигнатур вокруг пар (т.е. влево по массиву сигнатур и вправо от исходной), был разработан класс **SignatureComparator** (<https://github.com/Geridok/Player/blob/main/TestMovieListLibrary/SignatureComparator.h>)

Данный класс будет использоваться для поиска совпадающих видеофрагментов из видеопотока с роликами из базы данных. На данный момент ролики выбираются вручную. В класс **SignatureComparato**r передается порог (threshold) и, если сравнение сигнатур показало ниже результат чем порог, начинает поиск “влево” (compareLeft) и “вправо” (compareRight), от данного кадра для нахождения совпадающего видеофрагмента. Поиск, compareRight и compareLeft, планируется сделать многопоточным, однако при попытке сделать это с помощью стандартных методов (класса “thread”), было выяснено что внутри реализации вычисления сигнатур, используются библиотеки MFC, для понимания который необходимо больше времени, в силу отсутствия опыта работы с ней. Поэтому реализация многопоточности будет выполнена в одном стиле в следующем семестре.

Для тестирования скорости работы алгоритма, а так же определения точности работы алгоритма, был разработан графический интерфейс:



, а также набор классов, позволяющие выводить результаты и обрабатывать их в Exel. На представленном скриншоте есть поля, которые отображаю:

* Среднее время вычисление сигнатуры для первого видео (Average Time of calculating (First)),
* Среднее время вычисление сигнатуры для второго видео (Average Time of calculating (Second)),
* Среднее время выполнения сравнения сигнатур (Average Time of Comparing).

Range of Frames показывает номера кадров в исходном видео (в будущем поток прямого вещания), в которых был найден искомый ролик. Для этого часть ролика “Mersedes”, было смонтировано внутрь ролика “S7” в промежуток начиная примерно с 3-ей секунды. Данную часть ролика “Mersedes”, чуть больше 4 – ёх секунд и предполагалось найти. Далее полученный диапазон был сравнён вручную с исходными роликами и показал полное совпадение. Однако порог сравнения сигнатур пришлось подбирать вручную, соответственно вопрос как его подбирать остается открытым.

# Итоги

В ходе работы был проведен лит обзор по существующим методам реализации вычисления сигнатур, а также части из них экспериментального сравнения. Так же был разработан алгоритм сравнения пар сигнатур, а также сравнения вокруг пар (для поиска совпадающих видеофрагментов), который показал высокую точность, однако порог разницы сигнатур (threshold), необходимо было подбирать вручную, для получения высоких результатов. Была осознана архитектура итогового решения и разработаны классы с учетом этого (в лучших традициях ООП). Были определены горизонты распараллеливания разработанного алгоритма. Было разработано тестовое приложение, которое позволяет выводить скорость и результаты работы алгоритма, а также отображает их в пользовательском интерфейсе. Разработанное приложение позволяет выполнять поиск интересующего фрагмента в видео и отображает полученные результаты.

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

подпись студента расшифровка подписи

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы студента за отчетный период с «\_\_\_»\_\_\_\_20\_\_\_г. по «\_\_\_»\_\_\_\_20\_\_\_г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

подпись научного руководителя расшифровка подписи

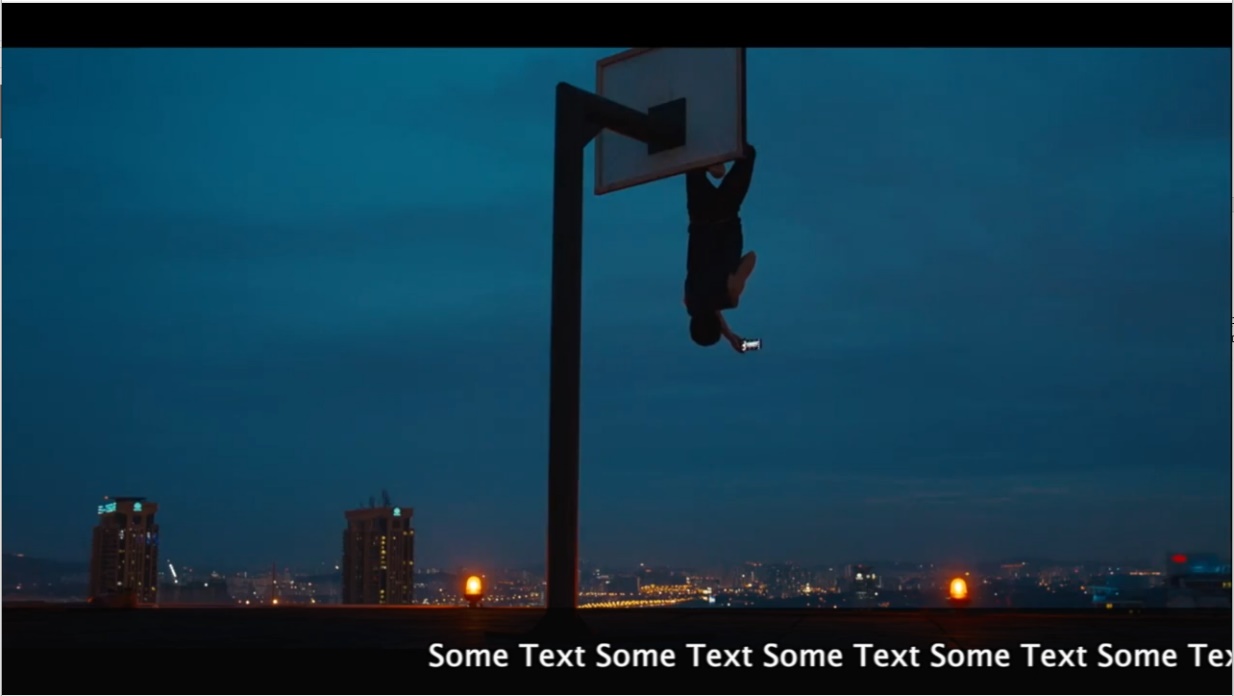
Оценка работы студента за отчетный период с «\_\_\_»\_\_\_\_20\_\_\_г. по «\_\_\_»\_\_\_\_20\_\_\_г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

подпись заведующего кафедрой расшифровка подписи

# Приложение

В основной части работы были представлены результаты работы алгоритмов SSIM и PSNR на примере вставки в видео большого заголовка. Так же было интересно посмотреть работу алгоритма при вставке бегущей, ведь это является частым случаем на телевидении.

Кадр с бегущей строкой

Кадр из исходного видео

# Список литературы

* + 1. ROBUST VIDEO HASH EXTRACTION Baris Coskun, Bulent SankurElectrical and Electronic Engineering Department, Bo azici University, Bebek, Istanbul
    2. Xavier Naturel and Patrick Gros., “A Fast Shot Matching Strategy for Detecting Duplicate Sequences in a Television Stream”., ACM., 2005.
    3. Поиск коротких последовательностей полноэкранных изображений в видео потоке, Ромб Евгений Олегович, Квалификационная работа степени магистра, г. Новосибирск, ИАиЭ СО РАН, кафедра АФТИ, 2008 г.