МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №14

ОТЧЕТ

ЗАЩИЩЕН С ОЦЕНКОЙ ПРЕПОДАВАТЕЛЬ

|  |  |  |
| --- | --- | --- |
| ассистент |  | А.В. Борисовская |
| должность, уч. степень, звание | подпись, дата | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1 |
| ИЗУЧЕНИЕ ФОРМАТА AVI |
| по курсу: МУЛЬТИМЕДИА ТЕХНОЛОГИИ |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |
| --- | --- | --- |
| СТУДЕНТ ГР. | 5721 | Михайлов Г.А. |
|  | подпись, дата | инициалы, фамилия |

Санкт-Петербург 2020

# Постановка задачи

## Цель работы

▪ Ознакомление со структурой файла в формате AVI

▪ Анализ статических свойств видеопоследовательностей

▪ Получение практических навыков работы с видеопоследовательностями

## Общее задание

1.2.1 Построить график автокорреляционной функции для трех видео, аргументом которой является временной интервал.

### 1.2.2 Сформировать новое видео в формате AVI, которое будет содержать кадры входного видео в обратном порядке;

1.2.3 Сформировать новое видео в формате AVI, которое содержит кадры первого входного видео, а за ними кадры второго.

## Индивидуальное задание (2с)

Изменение размеров исходного кадра в 2 раза по ширине и высоте.

# Описание структуры формата AVI

## Описание формата AVI RIFF

Формат AVI (Audio Video Interleave) используется для хранения видео, как совокупности аудиоданных и последовательности изображений. В качестве контейнерного AVI использует формат RIFF (Resource Interchange File Format), разработанный в 1991 году совместно компаниями IBM и Microsoft для функционирования на вычислительных машинах, использующих в представлении чисел порядок от младших к старшим байтам (little-endian). Поэтому внутри файла все константы имею именно такой порядок представления.

AVI-файлы могут содержать несколько потоков с различными данными, но большинство содержат два — один аудио и один видео. В данной работе будет рассматриваться простейший вариант AVIфайла, содержащий только видео поток.

## Основные элементы формата RIFF

Элемент формата RIFF начинается с четырехбуквенного кода, идентификатора элемента. За идентификатором следует двойное слово, определяющее длину данных, которые содержатся в элементе. За ними идут байты данных, относящихся к элементу. Обратите внимание, что число байт должно быть выравнено по границе слова, т.е. к данным, содержащим нечетное количество байт, в конце необходимо добавить один байт, значение которого неважно. Всего имеется два основных типа элементов.

▪ Порция (chank)

Синтаксис:

ckID ckSize ckData

где ckID — FOURCC-идентификатор,

ckSize — размер порции в байтах, ckData — данные, относящиеся к порции, если они есть.

▪ Список (list)\newline

Синтаксис:

’LIST’ listSize listType listData

где ’LIST’ — четырехбуквенный код, идентифицирующий элемент, как список, listSize — размер списка в байтах,

listType — FOURCC код, интерпретируемый как имя списка,

listData — данные, представляющие собой последовательную запись элементов RIFF: порций или списков.

Помимо перечисленных типов следует выделить заголовок файла в формате RIFF. Это порция, но ее синтаксис аналогичен списку, за исключением того, что вместо четырехбуквенного кода ’LIST’ используется ’RIFF’:

’RIFF’ fileSize fileType aviData

где fileType является четырехбуквенным кодом ’AVI’, а под aviData понимаются остальные данные файла.

## AVI RIFF

AVI файл идентифицируется FOURCC ’AVI ’ в RIFF заголовке. Все AVI файлы включают два обязательных списка, которые определяют формат потока и поток данных соответственно. AVI файл может включать порцию (chunk) индекса, которая описывает положение порций (chunks) данных в файле. Таким образом файл имеет следующую форму:

RIFF ('AVI '

LIST ('hdrl' ... )

LIST ('movi' ... )

['idx1' (<AVI Index>) ]

)

Первый обязательный список — ’hdrl’ описывает формат данных.

Второй обязательный список — ’movi’ содержит данные.

Список ’idxl’ содержит индекс. AVI файл должен содержать эти списки в правильной последовательности. Списки ’hdrl’ и ’movi’ использую вложенные порции (chunks) для представления данных.

Рассмотрим расширенную форму хранения списков в AVI RIFF:



## Основной заголовок AVI

Основной заголовок начинается со списка ’hdrl’, который содержит порцию (chunk) ’avih’. Основной заголовок хранит общую информацию об AVI файле, такую как количество потоков в файле и

ширину и высоту кадра. Основной заголовок может быть представлен структурой AVIMAINHEADER.

## Заголовки потоков AVI

Один или более списков ’strl’ следую за основным заголовком. Каждый поток требует ’strl’ списка, каждый такой список содержит информацию об одном потоке в файле и должен содержать порцию (chunk) заголовка потока ’strh’ и порцию (chunk) формата потока ’strf’. Дополнительно, ’strl’ может содержать порцию (chunk) данных заголовка потока ’strd’ и порцию (chunk) имени потока ’strn’. Порция (chunk) заголовка потока ’strh’ состоит из AVISTREAMHEADER структуры.

Порция (chunk) формата потока ’strf’ должна следовать за порцией (chunk) заголовка потока. Порция

(chunk) формата потока описывает формат данных в потоке. Данные, хранящиеся в этой порции (chunk), зависят от типа потока. Для видео потоков это структура BITMAPINFOHEADER, включая информацию о палитре, если необходимо. Для аудио потоков информация хранится в структуре

WAVEFORMATEX.

Если порция (chunk) данных заголовка потока ’strd’ присутствует, то она располагается непосредственно за порцией (chunk) формата потока. Формат и содержимое этой порции (chunk) описывается драйвером кодека. Обычно, драйверы используют эту информацию для конфигурации.

Приложениям, которые записывают или считывают AVI файлы, нет необходимости интерпретировать эту информацию. Они передают или получают эту информацию драйверу кодека как блок памяти.

Необязательная порция (chunk) имени потока ’strn’ содержит строку символов, заканчивающуюся нулем, описывающую поток.

Заголовки потоков в списке ’hdrl’ ассоциируются с потоками данных в списке ’movi’ в порядке появлений порции (chunk) ’strl’. Первая запись порции (chunk) ’strl’ ассоциируется с потоком О, вторая запись ассоциируется с потоком 1 и так далее.

## Поток данных (список ’movi’)

Список ’movi’ содержит данные потока — видео кадры и аудио отсчеты. Данные могут храниться непосредственно в списке ’movi’ или группироваться в списки ’гес ’. Список ’гес ’ группирует порции (chunks) так что бы они могли быть зачитаны с диска сразу (disk all at once). Иначе порции (chunks) хранятся с перемежением. Каждый FOURCC, описывающий порцию (chunks) данных, состоит из двузначного номера потока с двух символьного кода для определения типа информации в порции (chunk).

|  |  |
| --- | --- |
| Двух символьный код | Описание |
| db | Не сжатый видео кадр |
| dc | Сжатый видео кадр |
| pc | Смена палитры |
| wb | Аудио данные |

Например, если поток 0 содержит аудио, порция (chunk) данных для такого потока должна иметь FOURCC ’OOwb’.

Если поток 1 содержит видео, порция (chunk) данных для такого потока должна иметь FOURCC ’Oldb’ или ’Oldc’. Текстовые потоки могут использовать произвольные двух символьные коды.

## Индексы AVI

Необязательная порция (chunk) индекса ’idxl’ может следовать за списком ’movi’. Индекс содержит список порций (chunk) данных и их позиций в файле. Индекс состоит из структур AVIOLDINDEX для каждой порции (chunk) данных, включая порции (chunk) ’гес ’. Если файл содержит индекс, то в поле dwFlags структуры AVIMAINHEADER установлен флаг AVIF\_HASINDEX.

## Другие данные

Данные могут быть выравнены путем вставки порций (chunks) ’JUNK’. Приложения должны игнорировать содержимое этих порции (chunks).

# Описание работы с функциями библиотек для файлов AVI

Программный код был реализован на языке Python с использованием библиотеки AV, которая является обвязкой библиотеки FFMPEG, реализованной на языке C.

Также были использованы библиотеки:

▪ numpy – библиотека для выполнения математических действий;

▪ библиотека PIL с модулем Image, необходимая для работы с изображениями;

▪ opencv – библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения;

Методы для открытия и считывания файла:

*input\_container = av.open ('LR1\_1.AVI')*

Получить поток кадров можно получить следующим образом:

*stream = input\_container.streams.video[0]*

Функция, получающая из считаного файла пакеты данных:

*cur\_image = raw\_frame.to\_rgb().to\_image()*

Функция, позволяющая получить кадры из пакетов:

*input\_container.decode(stream)*

Функция, создающая новый поток с заданными параметрами:

*result\_stream = result.add\_stream(codec, rate=rate)*

Функция, кодирующая кадр в пакет:

*result\_stream.encode(image)*

Функция, складывающая пакеты в контейнер:

*result.mux (packet)*

Функция закрывает ранее открытый для записи контейнер:

*result.close()*

|  |  |  |
| --- | --- | --- |
| Кадры первого видео | | |
|  |  |  |
|  | Кадры второго видео |  |
|  |  |  |
|  | Кадры третьего видео |  |
|  |  |  |

Таблица 1 – Кадры исходных видео

# Результаты выполнения лабораторной работы

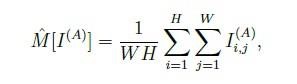
## Графики автокорреляции каждого кадра друг с другом

Для подсчета коэффициента автокорреляции мы циклом проходимся по матрице и для одного кадра считаем коэффициент автокорреляции с другим кадром, который мы сдвинули.

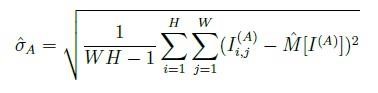
Для вычисления коэффициента автокорреляции используем формулу:

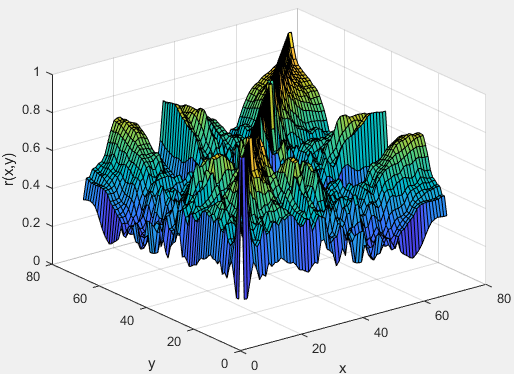
Для дополнительных расчетов коэффициента корреляции необходимы:

▪ Оценка математического ожидания



▪ Оценка среднеквадратичного отклонения





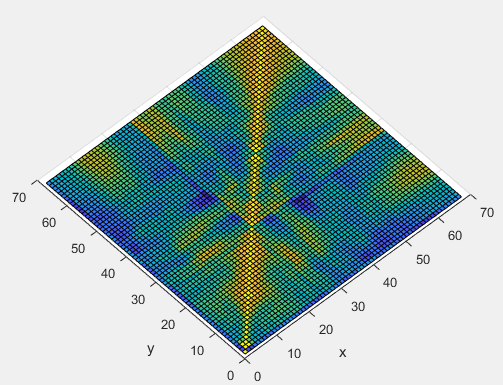
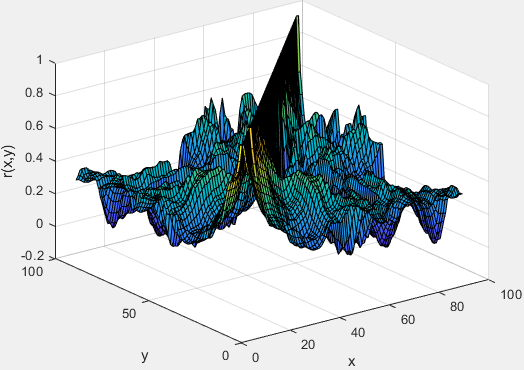


Рисунок 1 - График автокорреляции кадров для первого видео



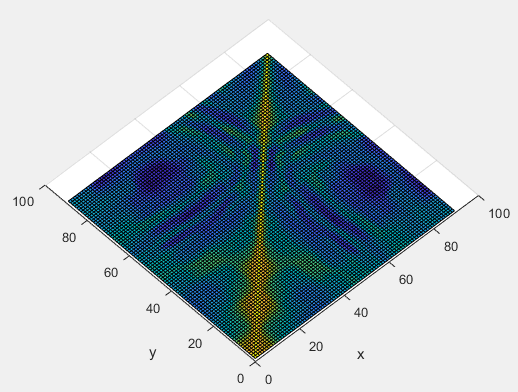


Рисунок 2 - График автокорреляции кадров для второго

видео



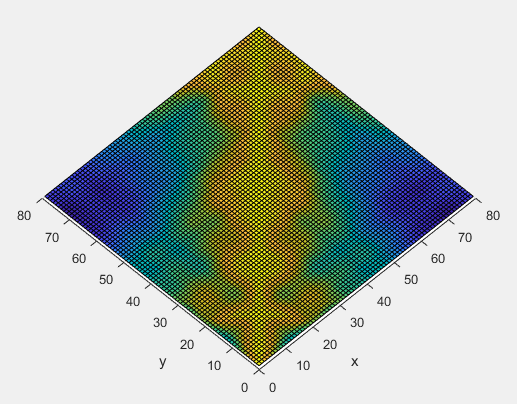


Рисунок 3 - График автокорреляции кадров для третьего

видео

Анализируя данный график можно сделать выводы:

* Для 1 и 2 видео коэффициент автокоррелирующей функции принимает значения от 0.2 до 1 и от 0 до 1. Из этого можно сделать вывод, что временная избыточность данных двух видео маленькая (кадры сильно «различаются» друг от друга) и сжатие этих видео не даст сильного эффекта.
* На графике автокоррелирующей функции для 3 видео мы видим, что временная избыточность данного видео высокая (статичный фон и кадры схожи друг с другом), поэтому эффективность сжатия данного видео будет высокая.

## Задание с перестановкой кадров

При реализации данного задания видеопоток был разбит на кадры и записан в массив, из которого они были перезаписаны в обратном порядке, а полученный массив записан в файл.

*result = av.open('out.avi', 'w')*

*result\_stream = result.add\_stream(codec, rate=rate)*

*result\_stream.height = height*

*result\_stream.width = width*

*result\_stream.pix\_fmt = format*

*for i in range(0,len(frames)):*

*image = av.VideoFrame.*

*from\_image(Image.fromarray(frames[len(frames)-i - 1], mode='RGB'))*

*for p in result\_stream.encode(image):*

*result.mux(p)*

*result.close()*

|  |  |
| --- | --- |
| Последний кадр в изначальном видео | Последний кадр после преобразования |
|  |  |

Таблица 2 – Результат работы кода

## Задание со слиянием двух видеофайлов

При реализации данного задания первый видеопоток был разбит на кадры и записан в массив. Аналогичное действие было проделано со вторым видеопотоком, записанным в конец того же массива. В результате полученный массив содержал кадры двух видеопотоков, следующих друг за другом. Далее был сформирован выходной файл.

*for fr in frames1:*

*image = av.VideoFrame.from\_image(Image.fromarray(fr, mode='RGB'))*

*for p in result\_stream.encode(image):*

*result.mux(p)*

*for fr in frames2:*

*image = av.VideoFrame.from\_image(Image.fromarray(fr, mode='RGB')) for p in result\_stream.encode(image):*

*result.mux(p)*

*result.close()*

|  |  |
| --- | --- |
| 1 | 2 |
|  |  |
| 3 | 4 |
|  |  |

Таблица 3 – Результат работы кода

## Индивидуальное задание

Изменение размеров исходного кадра в 2 раза по ширине и высоте

Открываем входной файл и записываем его в поток

*input\_container = av.open(filename)*

*stream = input\_container.streams.video[0]*

*height = stream.height*

*width = stream.width*

*format = stream.pix\_fmt*

*codec = stream.codec\_context.name*

*rate = stream.average\_rate*

Задаем новое значение размера кадра которое будем использовать

*new\_size = (width\*2, height\*2)*

Далее декодируем пакеты из потока на кадры

*input\_container.decode(stream)*

Используя библиотеку OpenCV, а именно ее функцию *resize* получаем кадры измененного размера заполняя появившиеся после расширения пиксели интерполированным по заданной области значениями

*res = cv2.resize(numpy.array(fr.to\_image()), new\_size, interpolation=cv2.INTER\_CUBIC)*

Значения интерполяции могут быть выбраны

* INTER\_NEAREST -интерполяция ближайших соседей
* INTER\_LINEAR - билинейная интерполяция (используется по умолчанию)
* INTER\_AREA -пересчет с использованием отношения площадей пикселей. Это может быть предпочтительным методом для децимации изображений, так как он дает муар ' - свободный результаты. Но когда изображение увеличено, оно похоже на то, что INTER\_NEAREST метод.
* INTER\_CUBIC -бикубическая интерполяция по окрестности пикселя 4x4
* INTER\_LANCZOS4 -интерполяция Ланцоса по окрестности 8x8 пикселей

Далее просто записываем полученные кадры в выходной файл

*result = av.open('out.avi', 'w')*

*result\_stream = result.add\_stream(codec, rate=rate)*

*result\_stream.height = height\*2*

*result\_stream.width = width \*2*

*result\_stream.pix\_fmt = format*

for fr in frames:

image = av.VideoFrame.from\_image(Image.fromarray(fr, mode='RGB'))

for p in result\_stream.encode(image):

result.mux(p)

*result.close()*

**

**

Таблица 4 – Результат выполнения программы

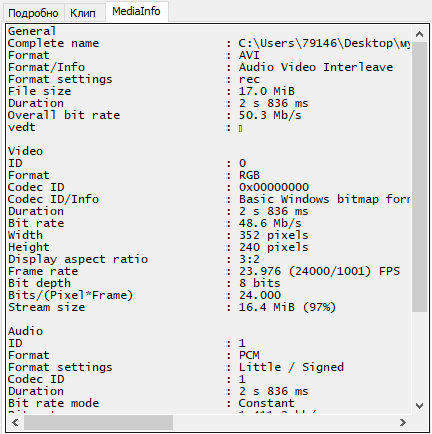
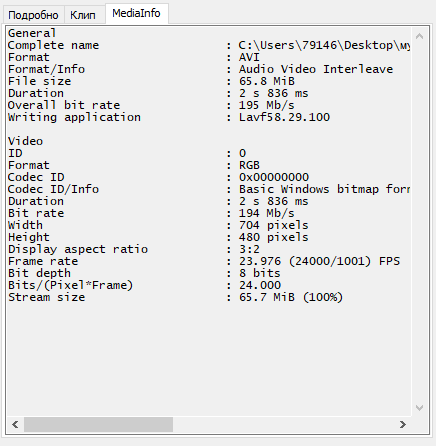
 

Таблица 5 – Свойства видео

# Выводы

В ходе выполнения лабораторной работы было осуществлено изучение строения и обработки видео формата AVI, а также библиотекой python AV для работы с такими файлами.

Были получены и проанализированы графики автокорреляции кадров для каждого видеофрагмента, после чего сделаны выводы по каждому графику.

По полученным коэффициентам автокорреляции были сделаны выводы, что чем больше значение коэффициента автокорреляции, тем больше временная избыточность в видео, поэтому данное видео можно сильно сжать. Так же можно сделать и обратный вывод, что чем меньше это значение, тем слабее будет эффект сжатия.

Также были изучены способы записи видео в обратном порядке, склейка кадров двух видео и изменение размеров кадра видео.

# Листинг программы

import math

import av

import numpy

import cv2

from av.video.stream import VideoStream

from PIL import Image

def rgb\_to\_y(pics, width, height):

y = []

for k, pic in enumerate(pics):

y.append([])

print(k)

for i in range(0, height):

y[k].append([])

for j in range(0, width):

blue, green, red = (pic[i\* height + j])

value = (int)(0.299 \* red + 0.587 \* green + 0.114 \* blue)

y[k][i].append(value)

return y

def expected\_value(y, a, height, width):

value = 0

for i in range(0, height):

for j in range(0, width):

value += y[a][i][j]

return (float)(value / (height \* width))

def sigma(y, a , height, width):

value = 0

expected\_val = expected\_value(y, a , height, width)

for i in range(0, height):

for j in range(0, width):

value += (y[a][i][j] - expected\_val) \*\* 2

return math.sqrt(value / (height \* width-1)), expected\_val

def correlation(y, a, b, sigma\_a, ev\_a, sigma\_b, ev\_b, height, width):

val = 0

for i in range(0, height):

for j in range(0, width):

val += (y[a][i][j] - ev\_a) \* (y[b][i][j] - ev\_b)

return (float) (val/ (width \* height \*sigma\_a \* sigma\_b))

def auto\_correlation(y, height, width):

correlation\_coeff = []

expected\_values = []

sigmas = []

for i in range(0, len(y)):

sig, expected\_val = sigma(y, i, height, width)

sigmas.append(sig)

expected\_values.append(expected\_val)

# вычисление мат ожидания и сигмы для каждого кадра

print("progress (" + str(len(y)) + ") = ", end =' ')

for a in range (0, len(y)):

correlation\_coeff.append([])

print(a,end=' ')

for b in range(0, len(y)):

value = correlation(y, a, b, sigmas[a],expected\_values[a] ,sigmas[b],expected\_values[b], height, width)

correlation\_coeff[a].append(value)

return correlation\_coeff

def fun1(filename):

array\_list = []

input\_container = av.open(filename)

input\_packets = input\_container.demux()

#print(input\_packets.stream.codec)

mas\_of\_pixels = []

width = 0

height = 0

for packet in input\_packets:

if isinstance(packet.stream, VideoStream):

# Получим все кадры пакета

frames = packet.decode()

for raw\_frame in frames:

cur\_image = raw\_frame.to\_rgb().to\_image()

mas\_of\_pixels.append(list(cur\_image.getdata()))

width, height = cur\_image.size

y = rgb\_to\_y(mas\_of\_pixels, width, height)

correl = auto\_correlation(y, height, width)

file = open('correlation3.txt', 'w')

min = 1

for i in range(0, len(correl)):

for j in range(0, len(correl[i]) - 1):

file.write(str(correl[i][j]) + ',')

if correl[i][j] < min:

min = correl[i][j]

file.write(str(correl[i][j]))

file.write('\n')

file.close()

print(min)

def fun2(filename):

input\_container = av.open(filename)

stream = input\_container.streams.video[0] #поток

height = stream.height

width = stream.width

format = stream.pix\_fmt

codec = stream.codec\_context.name

rate = stream.average\_rate

frames = []

for fr in input\_container.decode(stream):

frames.append(numpy.array(fr.to\_image()))

result = av.open('out.avi', 'w')

result\_stream = result.add\_stream(codec, rate=rate)

result\_stream.height = height

result\_stream.width = width

result\_stream.pix\_fmt = format

for i in range(0,len(frames)):

image = av.VideoFrame.from\_image(Image.fromarray(frames[len(frames)-i - 1], mode='RGB'))

for p in result\_stream.encode(image):

result.mux(p)

result.close()

def fun3(filename1, filename2):

input\_container = av.open(filename1)

stream = input\_container.streams.video[0]

height = stream.height

width = stream.width

format = stream.pix\_fmt

codec = stream.codec\_context.name

rate = stream.average\_rate

frames1 = []

for fr in input\_container.decode(stream):

frames1.append(numpy.array(fr.to\_image()))

input\_container = av.open(filename2)

stream = input\_container.streams.video[0]

frames2 = []

for fr in input\_container.decode(stream):

frames2.append(numpy.array(fr.to\_image()))

result = av.open('out.avi', 'w')

result\_stream = result.add\_stream(codec, rate=rate)

result\_stream.height = height

result\_stream.width = width

result\_stream.pix\_fmt = format

for fr in frames1:

image = av.VideoFrame.from\_image(Image.fromarray(fr, mode='RGB'))

for p in result\_stream.encode(image):

result.mux(p)

for fr in frames2:

image = av.VideoFrame.from\_image(Image.fromarray(fr, mode='RGB'))

for p in result\_stream.encode(image):

result.mux(p)

result.close()

def dop(filename):

input\_container = av.open(filename)

stream = input\_container.streams.video[0] #поток

height = stream.height

width = stream.width

format = stream.pix\_fmt

codec = stream.codec\_context.name

rate = stream.average\_rate

new\_size = (width\*2, height\*2)

frames = []

for fr in input\_container.decode(stream):

res = cv2.resize(numpy.array(fr.to\_image()), new\_size, interpolation=cv2.INTER\_CUBIC) #ресайз и интерполяция пустых пикселей

#INTER\_CUBIC -бикубическая интерполяция по окрестности пикселя 4x4

#INTER\_LANCZOS4 -интерполяция Ланцоса по окрестности 8x8 пикселей

frames.append(res)

result = av.open('out.avi', 'w')

result\_stream = result.add\_stream(codec, rate=rate)

result\_stream.height = height\*2

result\_stream.width = width\*2

result\_stream.pix\_fmt = format

for fr in frames:

image = av.VideoFrame.from\_image(Image.fromarray(fr, mode='RGB'))

for p in result\_stream.encode(image):

result.mux(p)

result.close()

def main():

#fun1('LR1\_1.AVI')

#fun2('LR1\_1.AVI')

#fun3('LR1\_1.AVI', 'LR1\_2.AVI')

dop('LR1\_1.AVI')

if \_\_name\_\_ == '\_\_main\_\_':

main()