# ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

## методические указания для выполнения лабораторных работ

М. Р. Гильмутдинов
Email: mgilm@vu.spb.ru

А.В. Чуйков
Email: alex@chkv.net
Государственный Университет Аэрокосмического Приборостроения
Санкт-Петербург

# 1 Изучение формата AVI

## 1.1 Цель работы

- Ознакомление со структурой файла в формате AVI
- Анализ статистических свойств видеопоследовательностей
- Получение практических навыков работы с видеопоследовательностями

## 1.2 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 имеют именно такой порядок представления.

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

### 1.2.1 Понятие четырехбуквенного кода FOURCC

FOURCC (**FOUR-C**haracter **C**ode) — это 32-хбитное беззнаковое целое из четырех конкатенированных (идущих последовательно один за другим) ASCII символов. Например, FOURCC 'abcd' представляет 0x61626364. FOURCC может содержать пробел, таким образом ' abc' (0x20616263) или 'abc ' (0x61626320) являются корректными четырехбуквенными кодами. FOURCC используются для идентификации внутренних элементов формата RIFF, что является удобным при поиске элементов во время просмотра RIFF-файла в обычном редакторе.

### 1.2.2 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 (chunk)
  Синтаксис:
  `ckID ckSize ckData`
  где `ckID` — FOURCC-идентификатор,
  `ckSize` — размер порции в байтах,
  `ckData` — данные, относящиеся к порции, если они есть.

- Список (list)
  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.

### 1.2.3 AVI RIFF формат

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 'idx1' 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:

```
RIFF ('AVI '
      LIST ('hdrl'
            'avih'(<Main AVI Header>)
            LIST ('strl'
                  'strh'(<Stream header>)
                  'strf'(<Stream format>)
                  [ 'strd'(<Additional header data>) ]
                  [ 'strn'(<Stream name>) ]
                  ...
                 )
             ...
           )
      LIST ('movi'
            {SubChunk | LIST ('rec '
                              SubChunk1
                              SubChunk2
                              ...
                             )
               ...
            }
            ...
           )
      ['idx1' (<AVI Index>) ]
     )
```

### 1.2.4 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.

### 1.2.5 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 0, вторая запись ассоциируется с потоком 1 и так далее.

### 1.2.6 Поток данных (список 'movi')

Как было сказан выше, список 'movi' содержит данные потока — видео кадры и аудио отсчеты. Данные могут храниться непосредственно в списке 'movi' или группироваться в списки 'rec '. Список 'rec ' 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 (описание опущено) |
| wb | Аудио данные |

Например, если поток 0 содержит аудио, порция (chunk) данных для такого потока должна иметь FOURCC '00wb'. Если поток 1 содержит видео, порция (chunk) данных для такого потока должна иметь FOURCC '01db' или '01dc'. Текстовые потоки могут использовать произвольные двух символьные коды.

### 1.2.7 Индексы AVI

Необязательная порция (chunk) индекса 'idx1' 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) 'rec '. Если файл содержит индекс, то в поле dwFlags структуры `AVIMAINHEADER` установлен флаг `AVIF_HASINDEX`.

### 1.2.8 Другие данные

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

## 1.3 Порядок работы с функциями Windows API для файлов AVI

Для работы с файлом в формате AVI используется специальная библиотека VFW (Video For Windows), входящая в состав Windows API. Прототипы функций и различные типы данных для работы с файлами AVI находятся в заголовочном файле `vfw.h`, который в свою очередь требует

подключения файла `windows.h`. Все функции, реализующие работу с файлами AVI, реализованы в системных библиотеках `msvfw32.dll`, `avifil32.dll`, `avicap32.dll`. Для того, чтобы компоновщик (linker) мог правильно сформировать вызовы указанных функций, к проекту необходимо в окне свойств компоновщика подключить библиотеку `vfw32.lib`. Информацию о том, как это сделать, можно получить в справочной системе MSDN.

Перед использованием функций работы с файлами AVI необходимо вызвать функцию `AVIFileInit()`. Если этого не сделать, все остальные функции работы с AVI будут возвращать код ошибки «библиотека AVIFile не инициализирована». При успешном выполнении большинство функций из библиотеки `vfw32.lib` возвращают нулевое значение.

Библиотеки для работы с AVI используют технологию COM (Component Object Model), поэтому функция `AVIFileInit` осуществляет также и инициализацию подсистемы COM. Далее осуществляется (подробно эти шаги описаны ниже):

- открытие/создание всех используемых файлов (входных и выходных)
- открытие входного видеопотока
- определение информации о потоке и формате кадров,
- создание выходного видеопотока, информация о потоке получена на предыдущем шаге
- задание формата кадров для созданного потока, формат был получен ранее
- чтение кадров из входного потока (входных потоков), их обработка и запись в выходной поток
- закрытие всех открытых потоков и файлов

Перед выходом из приложения следует вызвать функцию `AVIFileExit()`, которая производит освобождение занятой памяти и действия по завершению работы с COM.

### 1.3.1 Открытие файла AVI

При выполнении работы необходимо открыть файлы AVI, в зависимости от задания. Для открытия файла AVI используется функция `AVIFileOpen`:

```
STDAPI AVIFileOpen(PAVIFILE * ppfile, LPCTSTR szFile,
                   UINT mode, CLSID clsidHandler);
```

Кратко рассмотрим параметры функции. Объект «файл AVI» является объектом COM, реализующим интерфейс `IAVIFile`, поэтому для работы с ним необходимо иметь указатель на интерфейс. Этот указатель также называют дескриптором. Он имеет тип `PAVIFILE`, и указатель на дескриптор необходимо передать в качестве первого параметра функции. Второй параметр - это имя файла AVI. Третий параметр определяет режим работы с файлом, для данной работы следует использовать режим `OF_READ` при открытии входных файлов для чтения и режим `OF_CREATE | OF_WRITE` для создания нового выходного файла и записи в него результата. Четвертый параметр позволяет задать пользовательский обработчик, декодирующий сжатые кадры, но используется очень редко, обычно вместо него передают `NULL`. Функция открывает файл в выбранном режиме, и записывает полученный указатель на интерфейс `IAVIFile` в параметр `ppfile`. Возвращаемое значение равно нулю, если открытие файла прошло успешно, или содержит ненулевой код ошибки. Коды ошибок перечислены в MSDN или другой справочной системе. Таким образом, обычный вызов этой функции для открытия входных файлов может выглядеть так:

```
int result;
PAVIFILE avi;
result = AVIFileOpen(&avi, "C:\\MEDIA\\AVI\\Lab1_1.avi",
                     OF_READ, NULL);
if(result != 0)
    {Обработка ошибки}
```

### 1.3.2 Открытие потока видеоданных

Файл AVI в общем случае состоит из нескольких параллельных потоков (обычно из двух - видеопоток и аудиопоток). После того, как файл открыт, необходимо «открыть» поток, с которым будет идти работа. Для открытия существующего потока во входных файлах следует использовать функцию `AVIFileGetStream`:

```
STDAPI AVIFileGetStream(PAVIFILE pfile, PAVISTREAM * ppavi,
                        DWORD fccType, LONG lParam);
```

Первый параметр функции - это указатель на интерфейс `IAVIFile`, полученный при вызове функции `AVIFileOpen`. Поток также является объектом COM, реализующим интерфейс `IAVIStream`. Аналогично первому параметру функции `AVIFileOpen`, вторым параметром AVIFileGetStream передается указатель на указатель интерфейса. Третий параметр — тип потока (видео, аудио и т.д.), для видеопотоков следует использовать константу `streamtypeVIDEO` (типы потоков перечислены в MSDN). Четвертый параметр — это номер потока среди всех потоков указанного типа. Нумерация начинается с нуля. В данной работе входные файлы содержат всего один видеопоток, поэтому в качестве четвертого параметра следует передать 0. Функция открывает поток, и записывает указатель на созданный объект потока в параметр `ppavi`. Возвращаемое значение аналогично функции `AVIFileOpen`. Типичный вызов функции `AVIFileGetStream` выглядит так:

```
int result;
PAVISTREAM stream;
result = AVIFileGetStream(avi, &stream, streamtypeVIDEO, 0);
if(result != 0)
    {Обработка ошибки }
```

При работе с выходными файлами после создания файл пуст, поэтому видеопоток необходимо создать. Это можно сделать с помощью функции `AVIFileCreateStream`:

```
STDAPI AVIFileCreateStream(PAVIFILE pfile, PAVISTREAM * ppavi,
                           AVISTREAMINFO * psi);
```

Первые два параметра и возвращаемое значение этой функции аналогичны предыдущей, рассмотрим лишь третий параметр `psi`. Он является указателем на предварительно заполненную структуру `AVISTREAMINFO`:

```
typedef struct {
    DWORD fccType;
    DWORD fccHandler;
    DWORD dwFlags;
    DWORD dwCaps;
    WORD  wPriority;
    WORD  wLanguage;
    DWORD dwScale;
    DWORD dwRate;
    DWORD dwStart;
    DWORD dwLength;
    DWORD dwInitialFrames;
    DWORD dwSuggestedBufferSize;
    DWORD dwQuality;
    DWORD dwSampleSize;
    RECT  rcFrame;
    DWORD dwEditCount;
    DWORD dwFormatChangeCount;
    char  szName[64];
} AVISTREAMINFO;
```

Полное описание этой структуры можно найти в справочной системе MSDN. Большинство полей этой структуры можно заполнить нулевыми значениями (с помощью функции memset, например), поэтому рассмотрим только поля, которые используются в этой работе:

- `fccType` — тип потока, для видеопотоков следует использовать `streamtypeVIDEO`
- `fccHandler` — код декодера сжатых кадров, для несжатых кадров используется `BI_RGB`
- `dwScale`, `dwRate` — определяют количество кадров в секунду при воспроизведении. Эта величина равна `dwRate/dwScale`. Например, для 30 кадров/сек можно использовать `dwRate` = 30 и `dwScale` = 1, а для 24.97 кадров/сек используется `dwRate` = 25000 и `dwScale` = 1001
- `dwLength` — общее количество кадров в потоке
- `rcFrame` — структура `RECT`, хранящая информацию о размерах кадра в выходном видеопотоке

Типичный вызов функции `AVIFileCreateStream` можно реализовать так:

```
PAVISTREAM stream;
AVISTREAMINFO info;
memset(&info, 0, sizeof(info));
info.fccType    = streamtypeVIDEO;
info.fccHandler = BI_RGB;
info.dwScale    = 24;
info.dwRate     = 1;
info.dwLength   = 100;
SetRect(&info.rcFrame, 0, 0, 320, 240);
result = AVIFileCreateStream(avi, &stream, &info);
if(result != 0)
    {Обработка ошибки}
```

Функция `SetRect` — это функция Windows API, которая позволяет задать размеры прямоугольника, представленного структурой `RECT`. Ее описание можно найти в справочной системе.

### 1.3.3 Определение информации о потоке

После того, как входной поток открыт, для определения информации о нем следует вызывать функцию `AVIStreamInfo`:

```
STDAPI AVIStreamInfo(PAVISTREAM pavi, AVISTREAMINFO * psi,
                     LONG lSize);
```

Первый параметр - это указатель на интерфейс `IAVIStream`, полученный при вызове функции `AVIFileGetStream`, второй - указатель на структуру `AVSTREAMINFO` (рассмотрена выше), в которую будут помещены данные о потоке, третий — размер структуры, определяемый оператором `sizeof()`. Вызов выглядит так:

```
AVISTREAMINFO info;
AVIStreamInfo(stream, &info, sizeof(info));
```

Очевидно, что при создании потока эту функцию вызывать не требуется, так как структура `AVISTREAMINFO` заполняется еще при создании потока функцией `AVIFileCreateStream`.

### 1.3.4 Определение и задание формата кадров

Информация о формате данных каждого кадра хранится в структуре `BITMAPINFOHEADER`. Эта структура уже изучалась в курсе «Методы обработки аналоговой информации», поэтому лишь перечислим важные поля:

- `biSize` — размер структуры, должен быть равен `sizeof(BITMAPINFOHEADER)`

- biWidth, biHeight — размеры кадра
- biPlanes — количество плоскостей, равно 1
- biBitCount — бит на пиксель, в данной работе 24
- biCompression — сжатие, в данной работе BI_RGB
- biSizeImage — размер буфера, хранящего пикселы изображения

Для получения информации о формате данных потока используется функция AVIStreamReadFormat:

```
STDAPI AVIStreamReadFormat(PAVISTREAM pavi, LONG lPos,
                    LPVOID lpFormat, LONG * lpcbFormat);
```

В качестве параметров ей передаются: дескриптор IAVIStream, полученный при вызове функции AVIFileGetStream; позицию внутри потока, где находятся данные формата - обычно это 0; указатель на структуру BITMAPINFOHEADER, куда будет записана информация о формате; указатель на переменную, содержащую размер этой структуры, определяется как sizeof(BITMAPINFOHEADER). Возвращаемое значение равно нулю в случае успешного завершения, иначе ненулевой код ошибки. Типичный вызов:

```
BITMAPINFOHEADER bmpinfo;
LONG bmpinfosize = sizeof(bmpinfo);
AVIStreamReadFormat(stream, 0, &bmpinfo, &bmpinfosize);
```

Для задания информации о формате данных после создания потока используется функция AVIStreamSetFormat:

```
STDAPI AVIStreamSetFormat(PAVISTREAM pavi, LONG lPos,
                      LPVOID lpFormat, LONG cbFormat);
```

Список параметров у нее аналогичен функции AVIStreamReadFormat, но указатель на интерфейс IAVIStream получается при вызове функции AVIFileCreateStream (в общем случае указатель может быть получен и с помощью функции AVIFileGetStream, но это используется очень редко) и размер структуры передается как значение, а не указатель. Типичный вызов:

```
BITMAPINFOHEADER bmpinfo;
/* Заполнение структуры данными */
AVIStreamSetFormat(stream, 0, &bmpinfo, sizeof(bmpinfo));
```

### 1.3.5 Запись данных в поток и чтение из потока

Для чтения кадра из потока следует вызвать функцию AVIStreamRead:

```
STDAPI AVIStreamRead(PAVISTREAM pavi, LONG lStart,
                     LONG lSamples, LPVOID lpBuffer,
                     LONG cbBuffer, LONG * plBytes,
                     LONG * plSamples);
```

Рассмотрим параметры функции (применительно к видеопотоку, для аудиопотоков эти параметры имеют несколько иное значение): pavi — указатель на интерфейс IAVIStream, полученный при вызове функции AVIFileGetStream; lStart — номер первого кадра, считываемого из потока (или просто номер кадра, который нужно считать), нумерация начинается с нуля; lSamples — количество кадров, считываемых за данный вызов функции, для этой работы удобно использовать значение 1; lpBuffer — буфер в памяти, размером cbBuffer = biSizeImage (см. описание BITMAPINFOHEADER), куда будет записан кадр; остальные два параметра возвращают количество байт и кадров, считанных из потока, вместо них можно передать NULL. Возвращаемое значение аналогично всем остальным функциям для работы с AVI-файлами. Для записи в поток используется функция AVIStreamWrite:

```
STDAPI AVIStreamWrite(PAVISTREAM pavi, LONG lStart,
                      LONG lSamples, LPVOID lpBuffer,
                      LONG cbBuffer, DWORD dwFlags,
                      LONG * plSampWritten,
                      LONG * plBytesWritten);
```

Первые четыре параметра аналогичны предыдущей функции; `cbBuffer` — это размер буфера, для буфера под один кадр он равен `biSizeImage`; `dwFlags` для данной лабораторной следует указывать `AVIIF_KEYFRAME` — это означает, что кадр ключевой; оставшиеся два параметра аналогичны двум последним параметрам функции `AVIStreamRead`. Возвращаемое значение то же, что и у функции чтения из потока. Следует обратить внимание на то, что перед вызовом функций чтения/записи под буфер следует выделить память, например, с помощью функции `malloc`. Перед выходом из приложения эту память необходимо освободить, например, функцией `free`. Пример чтения и записи $i$-го кадра (код для обработки ошибок опущен):

```
void * buffer = malloc(bmpinfo.biSizeImage);
/*...*/
AVIStreamRead(stream1, i, 1, buffer, bmpinfo.biSizeImage, NULL, NULL);
AVIStreamWrite(stream2, i, 1, buffer, bmpinfo.biSizeImage,
               AVIIF_KEYFRAME, NULL, NULL);
/*...*/
free(buffer);
```

### 1.3.6 Закрытие потока и файла

По окончании работы с потоком и файлом их необходимо закрыть. Это делается с помощью функций `AVIStreamRelease` и `AVIFileRelease`. Они принимают единственный аргумент — указатель на соответствующий интерфейс. Примеры вызова:

```
AVIStreamRelease(stream);
AVIFileRelease(avi);
```

## 1.4 Требования к выполнению лабораторной работы

В ходе лабораторной работы необходимо:

1. Прочитать содержимое исходного одного или нескольких файлов с фильмом в формате AVI (Audio Video Interleave). Исходные файлы содержат только один поток видеоданных, кадры которого представлены в формате `RGB24` (24 бита на цвет).
2. Сформировать новый фильм в формате AVI, кадры в котором должны быть обработаны в соответствии с общими и индивидуальным заданиями.

Основными исходными данными для ЛР является файл `LR1_1.avi` и, если потребуется, файл `LR1_2.avi`.

### 1.4.1 Список общих заданий

1. Построить график автокорреляционной функции, аргументом которой является временной интервал (измеряемый в кадрах). Графики необходимо построить для следующих типов фильмов:
   (a) диктор,
   (b) медленное движение,
   (c) быстрое движение,
   (d) частая смена сцены.
2. Сформировать выходной файл, который будет содержать кадры входного фильма в обратном порядке.

3. Сформировать выходной файл, который содержит кадры первого входного фильма, а за ними кадры второго.

### 1.4.2 Список заданий

1. Изменение кадровой скорости
   (a) Уменьшение кадровой скорости (Frame Rate) с исходной в $N$ (целое) раз.
   (b) (*) Увеличение кадровой скорости в 2 раза. Дополнительные кадры формируются путем интерполяции соседних.
2. Изменение размеров изображения
   (a) Кадры выходного файла формируются как фрагменты кадров из исходного файла. Местоположение фрагмента определяется параметрами ($x$, $y$, $w$, $h$), где ($x$, $y$) являются координатами левого верхнего угла извлекаемого фрагмента, а ($w$, $h$) - размерами фрагмента. Координата (0, 0) находится в левом верхнем углу кадра. **Внимание!** При выводе на экран начало координат располагается в нижнем левом углу.
   (b) Выходной файл содержит копии кадров из первого фильма, но с модификацией пикселей, находящихся на расстоянии ближе, чем $N$ от края изображения. Необходимо реализовать эффект «плавного уменьшения» яркости по мере приближения к краю изображения.
   (c) (*) Изменение размеров исходного кадра в 2 раза по ширине и по высоте[1].
   (d) (*) Создать дополнительный поток типа текст, в который синхронно с каждым кадром помещать информацию о моменте его создания. Формат момента создания *“дд-мм-гггг:чч:мм:сс:мс”*.
3. Объединение двух файлов
   (a) Выходной файл содержит кадры, левая половина которых содержит модифицированный кадр из первого, а правая половина — модифицированный кадр из второго фильма. Модификация состоит в исключении из четных столбцов
   (b) Выходной файл содержит кадры, верхняя половина которых содержит модифицированный кадр из первого, и а нижняя половина - модифицированный кадр из второго фильма. Модификация состоит в исключении из четных строк
   (c) Выходной файл содержит кадры из первого фильма, перемежающиеся с кадрами из второго фильма по следующему принципу. Выходной кадр с номером $i$ является кадром из второго фильма, если $(i) \bmod N = 0$, в противном случае $i$-й является кадром из первого фильма (эффект “25-го кадра”).
   (d) (*) Выходной файл содержит кадры первого фильма, а за ним кадры второго. Между фильмами должен быть сформирован переход от кадра к кадру методом линейной интерполяции. Число кадров, участвующих в переходе равно $N$.
   (e) (*) Выходной файл содержит кадры из первого фильма. Внутрь каждого кадра при этом помещается уменьшенное в $N$ раз изображение из соответствующего кадра второго фильма.

### 1.4.3 Требования к оформлению отчета

Отчет должен содержать следующие пункты.

1. Постановка задачи.
2. Описание структуры формата AVI.
3. Алгоритм работы (чтение/запись) с файлом AVI с помощью функций Windows API.

[1] использовать алгоритм билинейной или бикубической интерполяции

4. Краткое описание используемых функций Windows API.
5. Графики автокорреляционной функции для нескольких типов фильмов (диктор/медленное движение/быстрое движение/частая смена сцены).
6. Алгоритм обработки последовательности видеокадров (в соответствии с индивидуальным заданием).
7. Листинг программы.

# 2 Исследование алгоритмов компенсации движения при обработке видеопоследовательностей

## 2.1 Обзор алгоритмов компенсации

Основным этапом компенсации движения при кодировании является процедура оценки движения, которая заключается в поиске для блока в текущем кадре наиболее «похожего» блока в базовом кадре. Поиск осуществляется в некоторой окрестности от места расположения блока, как показано на рис. **??**. Окрестность, в которой осуществляется поиск, как правило, является прямоугольником и характеризуется радиусом поиска по ширине и высоте, т.е. половиной ширины и высоты этого прямоугольника. Результатом процедуры оценки движения является нахождение вектора компенсации,

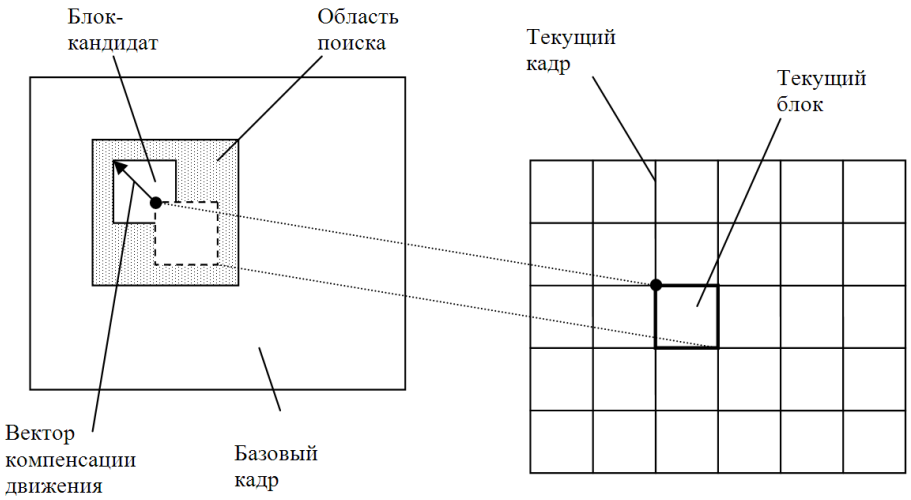


Рис. 1: Процедура оценки движения

указывающего положение наиболее «похожего» блока в базовом кадре относительно рассматриваемого блока в текущем кадре. Оценка движения выполняется для каждого блока в текущем кадре. Степень «похожести» блоков определяется с помощью специальных метрик, описание которых представлено ниже в соответствующем разделе.

### 2.1.1 Используемые обозначения

В лабораторной работе используются следующие обозначения:

1. $(\Delta x, \Delta y)$ — вектор компенсации движения, где $\Delta x$ — определяет смещение по оси абсцисс, а $\Delta y$ — по оси ординат. Как правило, вектор компенсации применяется к одному блоку внутри кадра, однако существуют варианты, когда это не так. Примером такого случая является глобальная компенсация движения, когда вектор компенсации применяется ко всему кадру в целом.

2. $(R_x, R_y)$ — радиус поиска в базовом кадре. Наиболее распространен случай, когда $R_x = R_y = R$.

3. $P^{cur}_{i,j}$ — пиксел, относящийся к текущему кадру. Индексы $i$ и $j$ могут задаваться относительно кадра в целом или относительно рассматриваемого блока в кадре.

4. $P^{base}_{i+\Delta x, j+\Delta y}$ — пиксел, относящийся к базовому кадру. Координаты пиксела в базовом блоке зависят от вектора компенсации движения.

5. $P^{diff}_{i,j}$ — пиксел, относящийся к разностному кадру. Индексы i и j могут задаваться относительно кадра в целом или относительно рассматриваемого блока в кадре.

6. $|x|$ — модуль числа $x$.
7. $|\{x\}|$ — мощность множества $\{x\}$.
8. $W_b, H_b$— ширина и высота блока, для которого выполняется процедура компенсации движения.

### 2.1.2 Метрики оценки «похожести» блока

1. Сумма абсолютных разностей

$$L_1 = \sum_{j=0}^{H_b-1} \sum_{i=0}^{W_b-1} \left| P_{i,j}^{cur} - P_{i+\Delta x, j+\Delta y}^{base} \right| \tag{2.1.1}$$

2. Энергия ошибки

$$L_2 = \sum_{j=0}^{H_b-1} \sum_{i=0}^{W_b-1} \left( P_{i,j}^{cur} - P_{i+\Delta x, j+\Delta y}^{base} \right)^2 \tag{2.1.2}$$

3. Функция ранжированных разностей. Введем в рассмотрение множество $\left\{ \tilde{P}_{i,j} \right\}$, элементами которого являются абсолютные разности, удовлетворяющие условию:

$$\left\{ \tilde{P}_{i,j} \right\} : \left| P_{i,j}^{cur} - P_{i+\Delta x, j+\Delta y}^{base} \right| < Thr \tag{2.1.3}$$

где $Thr$ — некоторый порог. Тогда функция ранжированных разностей $F_{rng}(Thr)$ определяется следующим образом:

$$F_{rng}(Thr) = \left| \left\{ \tilde{P}_{i,j} \right\} \right| \tag{2.1.4}$$

### 2.1.3 Поиск наиболее похожего блока методом полного перебора

Формально процедура поиска состоит из следующего алгоритма.

- Для текущего блока, координаты пиксела в верхнем левом углу которого равны $(x, y)$, ищется пиксел с такими же координатами в базовом кадре. Этот пиксел объявляется центром поиска.
- Строится прямоугольная область со сторонами $2R_x + 1$ и $2R_y + 1$ таким образом, чтобы центр поиска находился в центре этой области. Построенный прямоугольник будем называть *областью поиска*.
- Блоки-кандидаты выбираются на основе одного из пикселов, входящих в область поиска, так, чтобы этот пиксел находился в верхнем левом углу блока-кандидата. Т.е. множество пикселов, входящих в область поиска, определяет множество блоков-кандидатов, и мощности этих множеств равны.
- Следует отметить, что сам блок-кандидат не обязательно должен полностью находиться в области поиска. Достаточно, чтобы выполнялось правило выбора из предыдущего пункта.
- При полном переборе производится сравнение текущего блока с каждым блоком-кандидатом. Сравнение производится по одной из метрик оценки «похожести» блоков. Среди всех блоков-кандидатов выбирается один, который является наилучшим с точки зрения используемой метрики. Вектором компенсации движения $(\Delta x, \Delta y)$ будем называть смещение, которое определяет местоположение наилучшего блока-кандидата относительно координат текущего блока.
- На основе текущего блока и наилучшего блока-кандидата вычисляется разностный блок. Полученные вектор компенсации и разностный блок подвергаются дальнейшей обработке.

### 2.1.4 Подоптимальные алгоритмы поиска

Метод полного перебора является сложным с вычислительной точки зрения. Для ускорения процедуры поиска наилучшего блока-кандидата используют подоптимальные алгоритмы, которые рассматривают усеченное множество блоков-кандидатов. Платой за такое «ускорение» является вынесение неточного решения в случае, когда наилучший блок-кандидат не вошел в рассматриваемое множество. Несколько подоптимальных алгоритмов будут рассмотрены ниже.

#### Алгоритм монотонного поиска

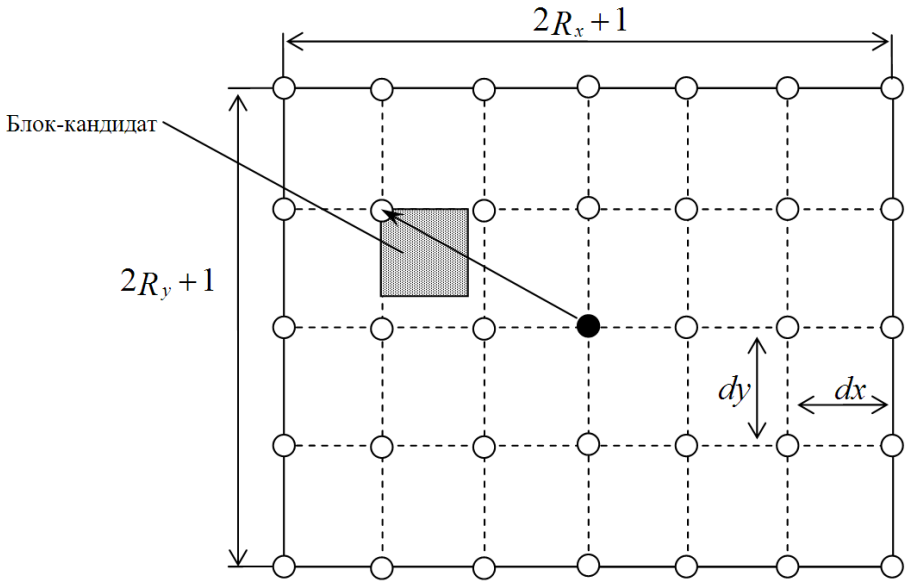


Рис. 2: Алгоритм монотонного поиска

На рис. **??** изображена схема, с помощью которой реализуется алгоритм монотонного поиска. В отличие от полного перебора, кандидатами на наилучший являются блоки, пиксел левого верхнего угла которых, обозначенных кружками. Кружками обозначены пикселы, которые находятся в левом верхнем углу блоков-кандидатов. Черным кружком обозначен левый верхний области поиска в базовом кадре. Расстояние между пикселами-кандидатами постоянно и составляет $dx$ пикселов по горизонтали и $dy$ пикселов по вертикали. Значения $dx$ и $dy$ являются параметрами алгоритма. Будем называть их *шагами*. Метод полного перебора является частным случаем монотонного поиска с шагом в один пикел.

#### Алгоритм поиска с «разбавленным» расстоянием

Алгоритм поиска с «разбавленным» расстоянием является модификацией алгоритма монотонного поиска. В отличие от последнего, область поиска в нем разделяется на внутреннюю и внешнюю, для каждой из которых реализуется алгоритм монотонного поиска со своим шагом, как это показано на рис. **??**.

### 2.1.5 Локализованный поиск

Локализованный поиск применяется для увеличения вероятности нахождения наилучшего блока-кандидата при использовании подоптимальных алгоритмов поиска. В силу того, что при подоптимальном поиске используется усеченное множество блоков-кандидатов, найденный наилучший среди них может оказаться не самым похожим блоков в заданном радиусе поиска. Пример такой ситуации изображен на рис. **??**. Здесь функция $F(x)$ в интерпретации поиска минимума для непрерывной функции $F(x)$. Поиск состоит из двух этапов. На первом этапе выполняется «грубый» монотонной

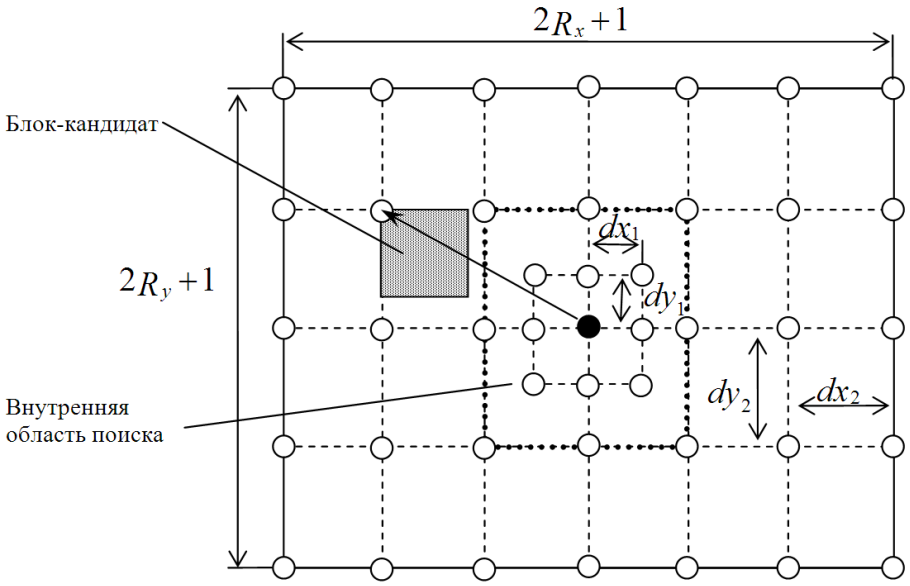


Рис. 3: Алгоритм поиска с «разбавленным» расстоянием

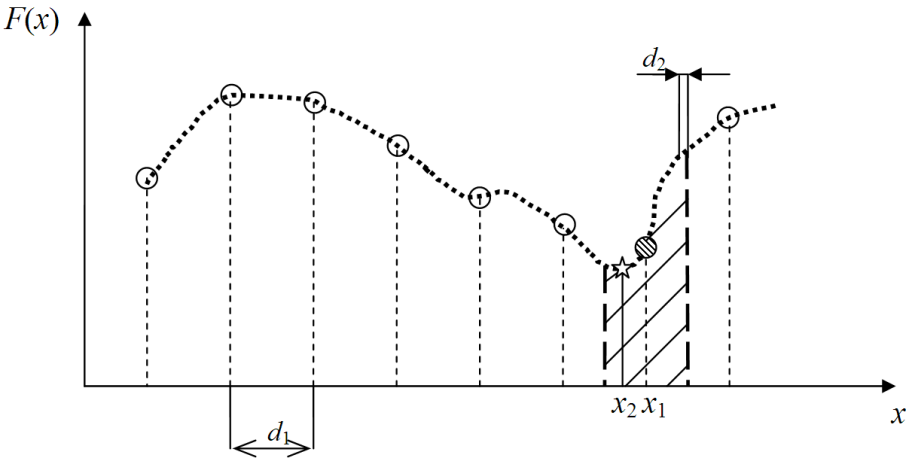


Рис. 4: Двухэтапный поиск минимума для непрерывной функции $F(x)$

поиск с шагом $d_1$, в результате которого наименьшим признается значение функции в точке $x_1$ (заштрихованный кружок). Однако фактическое наименьшее значение находится левее найденной точки, но в ближайшей ее окрестности. Наименьшее значение в точке $x_2$ (звездочка) может быть определено на втором этапе, в ходе которого осуществляется «точный» поиск в окрестности точки $x_1$ с шагом $d_2$. Преимущество такого метода состоит в увеличении вероятности нахождения более «похожего» блока-кандидата по сравнению с ситуацией использования подоптимального алгоритма. При этом вычислительная сложность поиска увеличивается незначительно в силу небольшого размера окрестности «точного» поиска на втором этапе. Как правило, локализованный поиск используется как дополнительный поиск при использовании подоптимального алгоритма.

## 2.2 Задание по лабораторной работе

В ходе выполнения лабораторной работы необходимо исследовать один из алгоритмов, относящихся к процедуре компенсации движения. В качестве входной последовательности кадров можно использовать кадры из фильмов *lr1_ 1.avi*, *lr1_ 2.avi* и *lr1_ 3.avi*. Все действия в лабораторной работе выполняются только над яркостной компонентой `Y`. Необходимо проделать следующие действия:

1. Выбрать один из вариантов индивидуальных заданий, представленных ниже.
2. Определить совместно с преподавателем радиус поиска для алгоритма компенсации движения.
3. Реализовать указанный в индивидуальном задании алгоритм компенсации движения. В качестве метрик, используемых при поиске блока в базовом кадре, необходимо использовать одну из метрик: $L_1$, $L_2$ или функцию ранжированных разностей. Требуемую метрику определяет преподаватель.
4. Предложить процедуру определения смены сцены в фильме.
5. Оценить энтропию сжатого потока для векторов компенсации движения.

Результатами работы являются:

- Файл *output.avi*, содержащий последовательность разностных кадров.
- Таблица оценки эффективности процедуры компенсации движения.

Кадры выходного фильма `output.avi` формируются на основе яркостной компоненты Y кадров входного фильма следующим образом.

1. Первый кадр входного фильма и первые кадры каждой сцены попадают в выходной файл без изменений.
2. Остальные кадры заменяются разностными, полученные в результате алгоритма компенсации движения, который соответствует индивидуальному заданию. Значение пиксела разностного кадра вычисляется по формуле:

$$P_{i,j}^{diff} = Clip\left(P_{i,j}^{cur} - P_{i+\Delta x, j+\Delta y}^{base} + 128, 0, 255\right) \tag{2.2.1}$$

   Здесь $Clip(x, a, b)$ обозначает операцию клипирования числа $x$ по диапазону $[a, b]$ (см. формулу ( **??**)). Таким образом, нулевая разность обозначается нейтральным серым цветом. Чем больше значение разности, тем более светлым будет пиксел при положительной разности, и темнее при отрицательной.

Таблица оценки эффективности процедуры компенсации движения оформляется в соответствии с таблицей **??**. Таблица состоит из четырех основных столбцов.

1. Номер кадра.
2. Значение дисперсии, рассчитанной для разностного кадра в целом.
3. Оценка энтропии для полученных векторов компенсации движения.
4. Тип кадра (сигнализирует о смене сцены).

### 2.2.1 Список индивидуальных заданий

1. Реализация и исследование подоптимальных алгоритмов поиска. Используются макроблоки $16 \times 16$.
   (a) Алгоритм монотонного поиска (с локализованным поиском)
   (b) Алгоритм поиска с «разбавленным» расстоянием (с локализованным поиском)
   (c) Алгоритм логарифмического поиска
2. Исследование компенсации движения для блоков различных размеров. Поиск осуществляется методом полного перебора в заданном радиусе. Для ускорения процедуры поиск необходимо в два этапа. На первом выполнять поиск для блока $16 \times 16$, на втором этапе осуществлять локализованный поиск для блоков требуемого размера с учетом вектора компенсации, полученного на первом этапе.
   (a) Использование макроблоков $16 \times 16$ пикселов.
   (b) Использование блоков $8 \times 8$ пикселов.
   (c) Использование блоков $4 \times 4$ пикселов.
   (d) Разбиение макроблока на блоки $8 \times 16$ или $16 \times 8$ пикселов.
   (e) Разбиение макроблока на блоки $8 \times 4$ или $4 \times 8$ пикселов.
3. Компенсация движения для B-кадров. Поиск осуществляется методом полного перебора в заданном радиусе.
   (a) Интерполяция базовых кадров.
   (b) Процедура выбора типа макроблока. При выборе ограничиться ситуациями предсказания по блоку в предыдущем кадре (предсказание назад) и в следующем кадре (предсказание вперед). При вычислении энтропии необходимо учитывать информацию о направлении предсказания.
   (c) Определение оптимального числа в группе кадров типа B.
4. Исследование дробнопиксельной компенсации движения. Поиск осуществляется методом полного перебора в заданном радиусе.
   (a) Полупиксельная компенсация с применением билинейной интерполяции.
   (b) Полупиксельная компенсация с применением бикубической интерполяции.
   (c) Четвертьпиксельная компенсация с применением билинейной интерполяции.
   (d) Четвертьпиксельная компенсация с применением бикубической интерполяции.
   (e) 1/8-пиксельная компенсация с применением билинейной интерполяции.
5. Глобальная компенсация движения.
   (a) Реализация алгоритма с использованием билинейной интерполяции.
   (b) Реализация алгоритма с использованием бикубической интерполяции.

Таблица 1: Таблица оценки эффективности процедуры компенсации движения

| *N* | *Дисперсия разностного кадра* | *Оценка энтропии* | *Тип кадра* |
|---|---|---|---|
| ... | ... | ... | ... |

### 2.2.2 Требования к оформлению отчета

Отчет должен содержать следующие пункты.

1. Постановка задачи
2. Общее описание алгоритмов компенсации движения
3. Алгоритм компенсации (в соответствии с индивидуальным заданием), результаты и графики. Выводы по результатам и графикам
4. Листинг программы