HIPER Найти

Основы использования утилиты AviSynth при обработке видео

1. Что такое AviSynth
2. Установка Avisynth
3. Чем составлять командные AVS файлы-скрипты
4. Основы языка скриптов
5. Открытие файлов-источников видео
6. Работа с чересстрочным видео
7. Фильтры деинтерлейса
8. Преобразование цветовых форматов
9. Коррекция яркости и цвета
10. Изменение размеров, обрезка, бордюры
11. Подавление шума
12. Сравнение действия фильтров
13. Повышение резкости
14. Подавление ореолов
15. Подавление радужных полос
16. Компенсация движения
17. Обработка звука
А еще что есть?

1. Что такое AviSynth

AviSynth (AVI-Синтезатор) — это очень полезная программная утилита, основанная на языке скриптов и включающая фильтры для простых (и не очень простых) задач нелинейной обработки видео. Она создана для детального доступа к видеокадрам клипов с возможностью производства над ними ряда хитрых манипуляций (по некоторому сценарию), недостижимых в традиционных монтажных программах (типа VirtualDub или Adobe Premier).

Что делает AviSynth уникальным по интерфейсу, так это факт, что он не является самостоятельной программой, имеющей графический интерфейс и производящей выходные файлы. Вместо того, AviSynth действует как «посредник» между видеофайлами и программами обработки видео, то есть как фрейм-сервер (кадр-сервер, податчик кадров).

Работа происходит следующим образом. Во-первых, вы создаете простой текстовый документ, так называемый скрипт (файл с расширением ***.AVS**), со специальными командами. Эти команды ссылаются на одно или более входное видео и на фильтры, которые вы хотите к ним применить. Затем вы запускаете видеоприложение (программу), например, VirtualDub, и открываете в нем файл скрипта. Тут начинает действовать AviSynth. Он открывает видеоисточники, на которые вы ссылались в скрипте, применяет указанные фильтры, и посылает результат видеоприложению. Приложение, однако, не знает, что это AviSynth работает в фоне. Вместо этого приложение думает, что оно напрямую открывает некий отфильтрованный AVI-файл, который располагается на вашем жестком диске.

Обычно выделяют пять главных причин, по которым пользователь приходит к AviSynth:

- AviSynth позволяет вам объединить вместе любое число видеофайлов. Вы можете даже выборочно объединять определенные части видео или заменять звуковые дорожки.
- Многие очень качественные фильтры видеообработки встроены в AviSynth (или в подключаемые модули-плагины). Например, фильтры для изменения размера, обрезки, подавления шума, повышения резкости, деинтерлейса.
- AviSynth может открыть почти любой тип видео, включая разные MPEG и Quicktime MOV. Однако когда AviSynth поставляет программе видео, для нее это выглядит подобно стандартному (несжатому) AVI. Это позволяет вам открыть некоторые форматы видео в программах, которые их не поддерживают.
- AviSynth генерирует видео, которое он посылает программе, порциями, на лету (в памяти). Следовательно, временных или промежуточных видеофайлов на диске не создается (раньше тема сохранения дискового пространства была более актуальной).
- Вы можете использовать AviSynth, чтобы открывать файлы, большие, чем 2 ГБ в некоторых старых программах, которые сами не поддерживают файлы такого размера.

По общему мнению, важнейшими достоинствами Avisynth являются:

- возможность работы в том цветовом формате, который наиболее подходит для входного или выходного видео, без лишних преобразований и с высокой скоростью.
- произвольный доступ к любому кадру и полю (половине кадра в чересстрочном видео), с возможностью их разделения

- произвольный доступ к любому кадру и полю (половине кадра в чересстрочном видео), с возможностью их разделения, прореживания, комбинирования, фильтрации, объединения, изменения частоты и т.п.

AviSynth включает все необходимые базовые инструменты для обработки видео, включая функции-источники для создания (чтения) клипов, фильтры для обработки изображения, фильтры для редактирования, фильтры для работы с чересстрочным видео, звуковые фильтры, специальные и экзотические фильтры, отладочные фильтры.

Это некоммерческая программа с открытым исходным кодом, начало разработки которой положил Бен Рудиак-Гоулд (Ben Rudiak-Gould), разработка которой продолжается группой энтузиастов в тесном контакте с пользователями.

При этом в дополнение к постоянно улучшаемым и шлифуемым базовым функциям, многими независимыми программистами разработано огромное число (до 200) дополнительных подключаемых модулей (плагинов). Эти плагины реализуют разнообразные функции, в том числе весьма сложные методы обработки для улучшения изображения (подавления шума и других дефектов), работы с полями чересстрочного видео и др. Принято деление на плагины, реализующие деинтерлейс и устранение Pulldown, пространственно-временное сглаживание, пространственное сглаживание, временное сглаживание, повышение/смягчение резкости, изменение размеров, субтитры, MPEG декодер, аудио декодер, сравнение качества видео, телевизионное вещание, разнообразные другие.

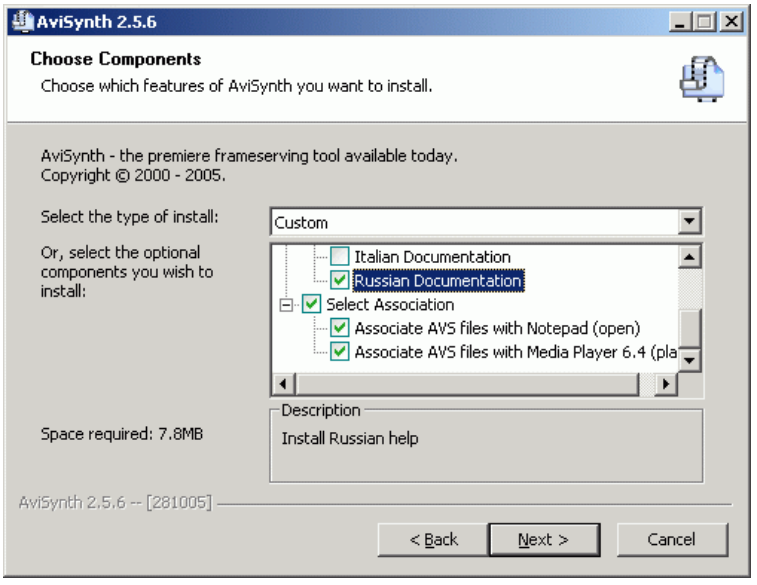
Путем комбинации различных фильтров в командном файле (скрипте), в том числе с функциями масочных операций и условными фильтрами, могут быть достигнуты наиболее эффективные методы обработки.

Большая часть разработчиков, ранее писавших плагины для редактора VirtualDub, как и большинство новых авторов, теперь пишут плагины только для AviSynth, так как это проще и быстрее для разработки (учтите бесплатность программ) и эффективнее по результатам. Поэтому пользователям, стремящимся получить наилучший результат, придется отчасти отходить от визуального интерфейса и осваивать язык AviSynth.

Многие хоть и наслышаны о существовании некоего "чудодейственного Ависинта", но не решаются на освоение, когда им советуют с его помощью решить некую задачу ("а может, это в Дубе сделать можно?"). Причин (проблем) три, и они тесно связаны. Во-первых, отпугивает отсутствие "кнопок", метод "тыка" по меню не работает, нужно писать команды вручную. Это хоть и ненастоящее, но все-таки программирование и необходимость думать. Во-вторых, выражения команд хоть и вполне вразумительны, но на английском языке, как и вся документация (весьма качественная, кстати). Эту проблему удалось решить в 2005 году, собрав команду и сделав официальный русский перевод к версии 2.5.6. Однако данный перевод представляет собой больше справочник по командам, а не руководство по использованию — когда, как, и какие скрипты и фильтры применять. Имеющиеся отрывочные сведения в руководствах по "рипанию" DVD очень фрагментарны, особенно на русском языке (да и на английском тоже). Это и является третьей проблемой, как бы оборотной стороной богатства возможностей и разнообразия функций AviSynth и плагинов. Действительно, богатейший опыт людей, отраженных в обсуждениях на форумах, не так-то просто использовать без некоего обобщения. Настоящая статья является попыткой помочь начинающим сделать первые шаги и приобрести навыки составления скриптов на примерах решения некоторых конкретных задач. Но не следует полагать, что всегда выручит слепое копирование приведенных скриптов без некоего осмысления. Приведенные скрипты несколько упрощены. Равно не следует ожидать, что кто-то на форуме тут составит оптимальный скрипт за Вас, и настроит все параметры, подходящие для конкретного видео. Сам себе не хочешь помочь — никто не поможет. Попробуйте, и творческий процесс вам понравится! (Автора занесло несколько дальше, и после нескольких лет ожидания момента, когда кто-то в мире напишет программу или фильтр для решения необходимой задачи, он сам взялся за это дело. Сейчас он, пожалуй, не скрипто-писатель, а плагино-писатель, поэтому в статье могут быть некоторые перекосы). Надеемся, опытным пользователям тоже кое-что из данного материала пригодится (ведь каждый — отчасти начинающий). Однако статья не представляет собой обзора всех возможностей AviSynth — напишите продолжение о том, в чем разбираетесь!

2. Установка Avisynth

Для использования Avisynth, его, как и другую программу, надо сначала установить в систему (поддерживаются Windows95 и выше). Загрузив дистрибутив последней стабильной версии (2.5.6a на момент написания статьи), запустите установку, не забыв отметить флажком опции установки английской и русской документации, как на рисунке.



После установки в системе будет в зарегистрирован тип файлов ***.AVS** и каталог автозагрузки плагинов типа "C:\Program Files\Avisynth 2.5\plugins". Документация будет доступна в меню **Пуск>Программы>Avisynth 2.5>Russian Avisynth Documentation** (не забудьте на досуге почитать раздел «о русском переводе»).

3. Чем составлять командные AVS файлы-скрипты

Наиболее естественным методом работы с Avisynth является прямое составление командных файлов-сценариев типа AVS, называемых скриптами, с помощью любого текстового редактора (Блокнот, UltraEdit, редакторы оболочек Total Commander, Far manager, и т.п.). Удобно пользоваться, в частности, подсветкой ключевых слов (например, с плагином Colorer для Far, файл синтаксиса avsColorer3 берите на моем сайте).

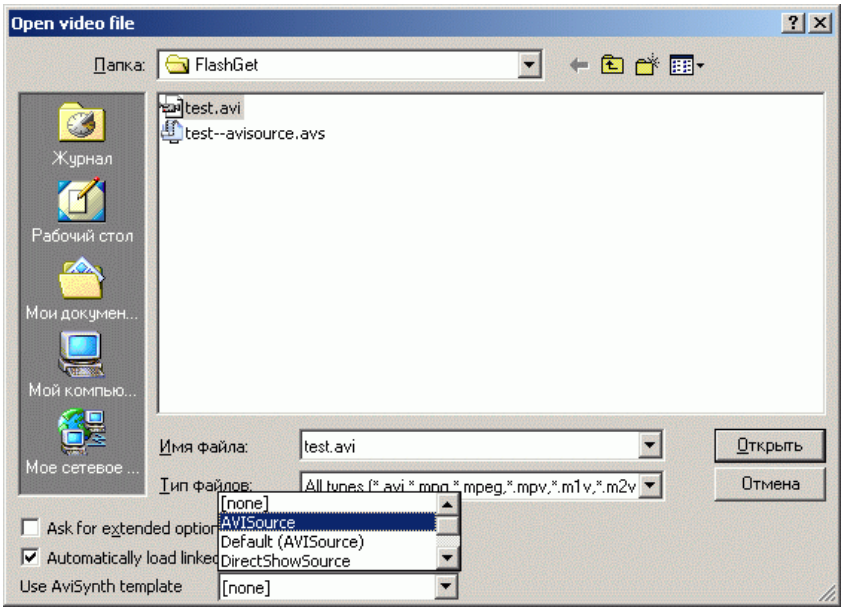
Однако именно необходимость ручного написания скриптов отпугивает новичков, хотя создание простого текстового файла не представляет собой какой-то сложности. Достаточно создать текстовый файл (с версии 2.5.7 появилась команда **Создать>AviSynth script** в меню Windows Explorer), написать в нем строку типа **Avisource("d:\file.avi")** и сохранить файл как Test.avs (если вы пользуетесь проводником Windows, то в настройках свойств папки снимите флажок с параметра "Скрывать расширения для зарегистрированных типов файлов").

Существует ряд программ, в которых сделаны попытки автоматизировать написание скриптов AVS для решения частных задач. Это пакеты для модного несколько лет назад перекодирования (рипа) DVD, такие, как AutoGordianKnot, GordianKnot, DVD2AVI, DVDRebuilder (AviSynth во многом развился для выполнения этих задач), MeGui.

Разрабатывались также несколько редакторов AVS скриптов: AvisynthEditor, AvsGenie, AviSynthesizer, SwiftAvs, DVX, AVSEdit, SyntEditor. Они имеют возможность генерации строк AVS скрипта путем выбора фильтров из меню, в том числе с настраиваемыми параметрами. Однако все эти редакторы не очень широко используются. Причинами этого, скорее всего, являются, во-первых, их некоторая недоделанность, а во-вторых, заторможенность (из-за использования медленных платформ типа VBasic, NET), и, в-третьих, фиксированность набора описаний известных программе фильтров (несмотря на возможность пополнения файлов описаний в формате XML, практически это не делается, так как требует непрерывной кропотливой работы ввиду постоянного обновления существующих фильтров и появления новых). Последнюю проблему может решить появившаяся в последних версиях AviSynth возможность автоматического опроса имен всех имеющихся функций и списка их параметров, что реализовано в бета-версии редактора avsFilmCutter. Частично это реализовано и в широко используемом и рекомендуемом (наряду с Блокнотом) редакторе VirtualDubMod, который представляет собой расширенный вариант известного видеоредактора VirtualDub (хотя и несколько устаревающий, ввиду замораживания разработки — может, кто из российских программистов подхватит?).

Для новичков, возможно, будет удобен VirtualDubMod с русифицированным интерфейсом, однако имеющийся перевод неофициальный и не вполне точный, поэтому в статье описан англоязычный вариант.

Первой полезной функцией VirtualDubMod является возможность автоматического создания скрипта AviSynth при открытии файла по шаблону фильтров-источников AviSource, DirectShowSource, Mpeg2Source (для файлов d2v).



Естественно, возможно открытие существующих файлов AVS-скриптов.
Касательно AviSynth, наиболее интересной возможностью VirtualDubMod является встроенный в него редактор AVS скриптов. Он вызывается из меню **Tools->Script Editor**.

Типичной задачей при монтаже является выбирание нужных частей видеоклипа и вырезание ненужных. В AviSynth вырезание производится командой TRIM(начало, конец). Естественно, для этого нужен визуальный контроль. Удобна следующая методика: щелчком на маркировочные стрелки выделяем нужную часть клипа в окне просмотра VirtualDubMod (как описано в многочисленных руководствах по VitrualDub), а затем в окне редактора скриптов используем команду меню **Edit-->Import Trim With Range** для вставки выделенного интервала в виде команд TRIM в то место строки, где стоит курсор (это удобно, если нужно отрезать края). Либо, наоборот, в окне просмотра VirtualDubMod выделяем и удаляем все ненужные отрезки (клавишей DEL), и затем в окне редактора скриптов используем команду меню **Import Frameset as Trims** для вставки набора нужных кадров без удаленных отрезков (это удобно если их много).

Также редактор скриптов позволяет закомментировать (превратить в неисполняемые) определенные строки, что удобно при отладке и подборе набора фильтров. Естественно, эту тривиальную операцию (добавление символа решетки **#** слева от команды в любой строке) можно сделать и руками. Все символы в строке после решетки считаются комментарием.
Необходимым для отладки скрипта является просмотр результата применения AviSynth фильтров. Нажатием клавиши **F5** достигается сохранение скрипта, его применение и обновление результата просмотра того кадра в окне VirtualDubMod. Более глубокое обновление (открытие скрипта заново) достигается клавишей **F7** (это необходимо при изменении числа кадров).

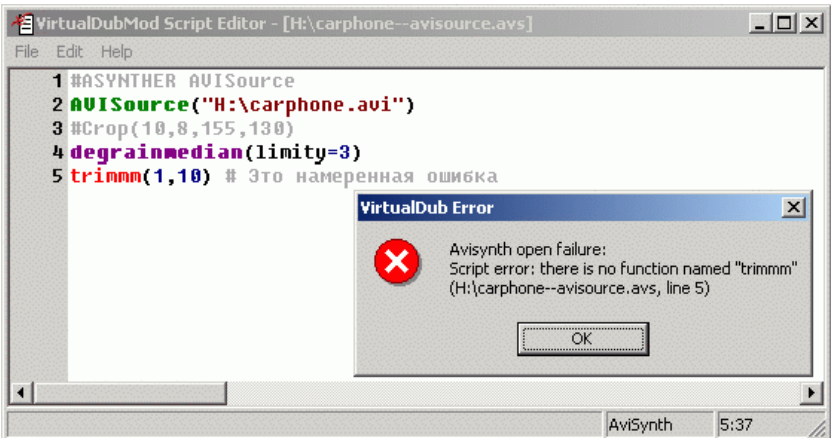
Еще одной автоматически формируемой командой является обрезка кадра **Crop**, значения формируются в зависимости от

примененной Вами обрезки (**Cropping**) выходного видео в окне VirtualDubMod.

Другие команды в окне редактора придется все же писать руками, поглядывая в документацию по Avisynth. (Дополнение: "Eudg" подсказал недокументированый способ автозавершения при вводе функций через Ctrl-пробел).

Доступные функции установленных внешних плагинов можно посмотреть в меню **Help>Avisynth Information**.

Прелестной особенностью редактора скриптов является подкрашивание ключевых слов. Базовые функции AviSynth подкрашиваются зеленым, функции плагинов — сиреневым, слегка неверный синтаксис подкрашивается красным, цвет меняется по мере набора символов. Названия параметров и их значения редактором не проверяются, поэтому возможны ошибки, о которых сообщает AviSynth при обновлении по **F5**. Пример окна редактора и сообщения об ошибке («нет функции Trimmm, строка 5 скрипта») приведены на рисунке. Исправьте ошибку и обновите вид. Сообщение об ошибке можно скопировать в буфер, нажав Ctrl-C (и вставить его в свое сообщение на форуме).



Рекомендуется к использованию разработанный недавно (написанный на языке Python и также свободно распространяемый по лицензии GNU GPL) редактор скриптов AvsP (автор Peter Jang "qwerpof") с автозавершением, подсветкой, подсказкой списка параметров, справки, ползунками настройки значений, встроенным окном просмотра, автоматической генерацией строки источников при перетаскивании мышкой, вкладками для нескольких скриптов, закладками на кадрах и многими другими удобствами, есть и русификация.

4. Основы языка скриптов

Так как словарь AviSynth состоит из более чем ста слов-функций, каждая из которых может имеет несколько параметров, описать их все, и тем более их возможные комбинации в скриптах, в настоящей статье невозможно (обращайтесь к документации).

А ведь есть еще и плагины, среди них бывает несколько (от разных авторов), реализующих подобную функцию, но отличающихся в деталях. Особенно много всевозможных денойзеров (подавителей шума). Почти каждый плагино-писатель, видимо, считает себя не реализовавшимся, пока не напишет несколько своих плагинов (автор статьи не исключение). Естественно, это объясняется и разнообразными видами шума. Тем не менее, качество и степень доводки плагинов разные. Некоторые написаны "на коленке" недоучившимся школьником за час (какие-то из них — настоящие жемчужины), а некоторые обновляются и совершенствуются последовательно несколькими людьми в течение лет. Описания наиболее полезных и проверенных плагинов включены в документацию по AviSynth. Но, поскольку пристрастия видео-сообщества со временем меняются, наиболее свежую информацию всегда можно найти на форумах. Там можно найти и разнообразные скрипты, написанные любителями видео в попытках решения своих задач.

Синтаксис скриптов AviSynth (подробно описанный в документации) допускает несколько способов написания, как прямое присвоение переменных, так и объектно-ориентированную запись функций (с точкой), в том числе с возможностью неуказания подразумеваемых переменных.

В скриптах AviSynth основной единицей обработки является видеоклип. Видеоклип является основным (первым, обычно неименованным) аргументом функции. Результатом выполнения функции также является видеоклип, который присваивается переменной с явно указанным именем или подразумеваемой переменной LAST (последний), которая может быть не указана, и является подразумеваемым аргументом следующей функции.

Большинство фильтров имеют дополнительные параметры, позволяющие настроить оптимальный режим обработки для конкретного видеоклипа, обычно все параметры имеют некоторые значения по умолчанию, так что можно указывать явно только те параметры, значения которых необходимо подстроить.

Во избежание недоразумений отметим, что Ваш файл на жестком диске не меняется при обработке, все кадры клипов создаются в оперативной памяти.

Для подключения в AviSynth дополнительных модулей (плагинов), нужно сначала их откуда-то скачать, затем извлечь из скаченного архива (обычно типа ZIP) файл типа DLL и добавить его в специальный подкаталог типа "C:\Program Files\Avisynth 2.5\Plugins" для автоматической загрузки, либо явно указывать в каждом файле скрипта команду загрузки каждого нужного плагина вида **LoadPlugin ("PluginFileName.DLL")**. Рекомендуем автозагрузку, но ниже во всех примерах скриптов для информации о необходимых плагинах указаны команды явной загрузки (их можно убрать, закомментировав). Часто имя функции совпадает с именем плагина, но не всегда, плагин может содержать и несколько функций. В архиве распространяемых плагинов обычно также есть написанные автором файлы документации (типа TXT или HTML), а также исходные тексты программы для возможности усовершенствования плагина другими программистами.

5. Открытие файлов-источников видео

Практически любой скрипт включает открытие файлов источников видео (и аудио). В базовый набор AviSynth входят фильтры:

Практически любой скрипт включает открытие файлов-источников видео (и аудио). В базовый набор AviSynth входят фильтры-источники для открытия AVI-видеофайлов **AVISource** (рекомендуется для классических AVI типа DivX и т.п. через интерфейс VfW), **DirectShowSource** (через интерфейс DirectShow, ввиду особенностей рекомендуется применять, если не работает **AVISource** или специализированный плагин требуемого формата), звуковых файлов **WavSource**, а также файлов изображений **ImageSource**. Для чтения ряда других форматов файлов разработаны (и продолжают разрабатываться) отдельные плагины, в том числе:

- RawSource (разработчик Ernst Peche "WarpEnterprises") для "сырых", несжатых видеоданных;
- QTSource (разработчик Josh Harris "tateu") для файлов QuickTime;
- FFMpegSource (разработчик "Myrsloik") для множества форматов, открываемых MPlayer;
- разрабатывается плагин DGAVCDec (разработчик Donald Graft на основе libavcodec.dll) для чтения AVC/H.264 (с позиционированием) и др.

Для файлов MPEG1 и MPEG2 в настоящее время наиболее отработанным является использование функции **Mpeg2Source** плагина DGDecode (пакет DGMPGDec, разработчик последних версий Donald Graft "neuron2"). Для этого предварительно необходимо воспользоваться программой DGIndex из пакета DGMPGDec, в которой открыть файл MPEG2 и сохранить вспомогательный индексный файл анализа типа D2V (обеспечивающий точное позиционирование при навигации и прокрутке), который и должен быть указан в качестве аргумента функции **Mpeg2Source** (одновременно выделенный звук должен быть загружен отдельной командой, см. раздел по обработке звука.)

```
# Пример скрипта 5.1
# Откроем DVD видео, указав файл индекса D2V
MPEG2Source(d2v="mydvd.d2v")
# Так как результат ничему не присваивается явно,
# то он присваивается в переменную LAST
```

Результатом выполнения скрипта, передаваемым в вызывающую программу, является либо последнее значение переменной LAST (то есть последний неименованный результат), как в примере 5.1, либо явно указанный для возврата клип (вида Return LAST, или Return A), как в примере 5.2.

```
#Пример скрипта 5.2
# Откроем несколько AVI видеофайлов, и объединим вырезки
# переменная A (английская) — клип из файла film1.avi
A = AviSource("D:\video\film1.avi")
# выделим из клипа A нужные кадры с 100 по 999
# и присвоим этот отрезок переменной A1
A1 = Trim(A, 100, 999)
# выделим из клипа A нужные кадры с 3000 по 4000
# и присвоим его переменной A2.
A2 = Trim(A, 3000, 4000)
# переменная B — клип из файла film2.avi
B = AviSource("D:\video\film2.avi")
# Выделим из клипа B первые 501 кадр (пример объектной записи)
B = B.Trim(0, 500)
# Объединим получившиеся три отрезка в результирующий клип
A1 ++ A2 ++ B
# Результат присвоился в переменную LAST
# Преобразуем видео в черно-белое (просто как пример фильтра)
GreyScale()
# Так как аргумент типа клип для функции не указан явно,
# то функция применяется к имеющемуся значению LAST
# Так как результат функции тоже ничему не присваивается явно,
# то он передается в переменную LAST
return LAST # возвращаем LAST
```

Если нужно объединить видео из нескольких входных файлов, то надо в скрипте для каждого файла написать команду открытия, с присвоением в разные переменные. Затем можно каждый клип последовательно подрезать (при необходимости отрезать рекламу, например), и результаты объединить. Удобно открыть первый клип, выполнить обрезку (при необходимости), закомментировать эти строки, затем открыть следующий клип, выполнить обрезку, закомментировать, и т.д. Затем убрать комментарии, добавить команды объединения (плюсы) и получить итоговый объединенный клип.

```
#Пример скрипта 5.3
# Откроем несколько AVI видеофайлов, и объединим вырезки
# переменная A (английская) — клип из файла film1.avi
A = AviSource("D:\video\film1.avi")
A # Помещаем клип A в LAST
A=Trim(100, 999) ++ Trim(3000, 4000) # набор кадров (Frameset)
# переменная B — клип из файла film2.avi
B = AviSource("D:\video\film2.avi")
B # Помещаем клип B в LAST
# Выделим из клипа B первые 501 кадр (пример объектной записи)
B = Trim(0, 500)
# Объединим получившиеся отрезки в результирующий клип
A ++ B
# результат ничему не присваивается явно, подразумевается LAST
```

Иногда спрашивают, как обратить видео (обратить ход времени) в качестве эффекта. С помощью AviSynth это делается очень просто, функцией **Reverse** (пока не касаемся вопросов звука).

#Пример скрипта 5.4

```
#Пример скрипта 5.4
# Эффект обращения времени
AviSource("D:\video\film1.avi") # входной видео файл
A = Trim(0, 100) # Отрезок до эффекта
B = Trim(100, 999) # Выделим некоторый отрезок для сцены эффекта
C = Trim(1000, 0) # остаток входного клипа
A ++ B ++ Reverse(B) ++ B ++ C
# Объединили отрезки в результирующий клип из 5 частей ( пример):
# 1. Часть видео до сцены эффекта,
# 2. сцена эффекта с нормальным ходом времени,
# 3. сцена эффекта с обратным ходом времени,
# 4. повтор сцены эффекта с нормальным ходом времени,
# 5. остаток видео до конца
```

6. Работа с чересстрочным видео

В кино в каждый момент времени снимается кадр целиком. В традиционном (чересстрочном) телевидении и видео (за исключением недавно появившихся прогрессивных стандартов) каждый кадр состоит из строк, образующих два поля, снятых в разные моменты времени. В один момент снимаются четные строки (первое, верхнее поле), в следующий момент — нечетные (второе, нижнее поле), и так далее. Однако кино часто показывают по телевидению (и записывают на DVD), в странах с разной системой телевидения и, соответственно, с разными частотами кадров, не совпадающими с частотой кадров кинофильма. В этом случае студии производят либо ускорение видео (с 24 до 25 Гц для PAL) либо повтор некоторых полей по определенному алгоритму (Pulldown) или (что хуже) смешение (blend). Обработку видео (например, фильтрацию) желательно вести в том виде, в котором оно существовало изначально.

Начнем с рассмотрения простого случая истинного чересстрочного видео, снятого видеокамерой. Наиболее естественным (для сохранения плавности движений) является сохранение чересстрочности при обработке, кодировании и записи DVD. Обработку видео целесообразно вести, разделив его на поля, или использовать функции-фильтры, имеющие опцию обработки видео как чересстрочного.

Avisynth имеет функцию **SeparateFields** для разделения кадров видео на поля (с удвоением частоты) и функцию **Weave** для их обратного сцепления в кадры.

```
#Пример скрипта 6.1.
# Применение пространственного фильтра
# к чересстрочному видео, разделенному на поля
LoadPlugin("C:\Plugins\VagueDenoiser.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
SeparateFields() # разделяем видео на поля
VagueDenoiser(threshold=2)
# фильтруем каждое поле пространственным фильтром
Weave() # соединяем поля в кадры
```

В данном примере 6.1 мы использовали для снижения шума один из лучших пространственных фильтров (функция **VagueDenoiser**), который исполнен как плагин (файл VagueDenoiser.dll, разработчики "Lefungus", "Kurosu", "Fizick"), и поэтому сначала должен быть загружен (автоматически или явно, наглядней в начале скрипта). Сила (порог) фильтрации задан параметром threshold=2, остальные параметры со значениями по умолчанию.

Еще обратите внимание на такую мелочь, что в функциях без аргументов мы пишем пустые скобки (работать фильтр будет и без них, но без кэширования, медленнее — это будет исправлено с версии AviSynth 2.5.7).

```
#Пример скрипта 6.2
# Применение к чересстрочному видео пространственного фильтра,
# имеющего опцию работы чересстрочно (interlaced)
LoadPlugin("C:\Plugins\VagueDenoiser.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
# фильтруем каждое поле пространственным фильтром,
# включив опцию чересстрочности
VagueDenoiser(threshold=2, interlaced=true)
```

Более сильную фильтрацию обеспечивают временные фильтры, учитывающие информацию и о соответствующих точках в соседних кадрах, или пространственно-временные, рассматривающие соседние точки в пространстве и во времени.

Так как строки содержат информацию о сдвинутых по вертикали на одну строку точках, то после разделении на поля информация в соответствующих строках не относится к тому же месту пространства, и не должна быть смешена. Поэтому из потока полей выделяют поток четных полей и поток нечетных полей, в каждом из которых информация относится к соответствующим (несмещенным) точкам.

```
#Пример скрипта 6.3
# Применение пространственно-временного фильтра
# к чересстрочному видео
с разделением на четные и нечетные поля
LoadPlugin("C:\Plugins\DeGrainMedian.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
SeparateFields() # разделяем видеокадры на поля
# Выделяем четные (0,2,4…) поля в переменную Even
Even = SelectEven()
# фильтруем четные поля и замещаем Even результатом
Even = DeGrainMedian(Even)
# Выделяем нечетные (1,3,5,…) поля в переменную Odd
```

```
Odd = SelectOdd()
# фильтруем четные поля и замещаем Odd результатом
Odd = DeGrainMedian(Odd)
Interleave(even, odd) # чередуем четные и нечетные поля,
# собирая исходную последовательность (0,1,2,3,4,5…)
Weave() # соединяем поля в кадры
```

Некоторые фильтры, в том числе используемый тут DeGrainMedian, имеют опцию обработки чересстрочного видео, при этом они реализуют этот же алгоритм раскладки на поля внутренне, в этом случае скрипт упрощается.

```
#Пример скрипта 6.4
# Применение к чересстрочному видео пространственно-временного
# фильтра,
имеющего опцию работы с чересстрочным видео
LoadPlugin("C:\Plugins\DeGrainMedian.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
DeGrainMedian(interlaced=true) # фильтруем чересстрочно
```

7. Фильтры деинтерлейса

Деинтерлейс применяют для удаления эффектов «гребенки» чересстрочного видео для комфортного просмотра на мониторе компьютера (на телевизоре эта гребенка будет не видна). Можно выделить случай, когда видео изначально было прогрессивным (фильмом), для преобразования в видео было применено некое чередование полей (telecine, pulldown), и нам необходимо выделить из видео последовательность оригинальных кадров. При этом имеется в виду, что смешения полей нет, и вся трудность просто в нахождении нужных. В простом случае фильма в NTSC можно воспользоваться базовой функцией **Pulldown**. Можно использовать и детально использовать богатые возможности AviSynth по разложению на поля, их ручном анализе, и составлением некоторого скрипта для их комбинации. В более сложных реальных случаях для автоматизации процесса целесообразно применять плагины **DeComb** (разработчик Donald Graft) или **TIVTC** (разработчик "Tritical"), реализующие методы подбора полей (field matching) из нескольких соседних кадров для восстановления прогрессивных кадров, в основном для формата NTSC, в том числе из гибридного видео и др. Оба плагина состоят из двух главных функций, на первом этапе осуществляющих подбор подходящих полей и формирование кадров, а на втором этапе — отбрасывание ненужных дубликатов (децимацию). Между собой они могут общаться с помощью подсказок, спрятанных в кадре. Плагин **TIVTC** активно разрабатывается в настоящее время и имеет несколько больше возможностей. Основными (базовыми) параметрами его функции **TFM** являются:

- "order" — порядок полей (0 — BFF, 1 — TFF, -1 = из AviSynth, по умолчанию);
- "mode" — режим от 0 до 6, грубо чем больше, тем больше соседних полей пробуется, и еще специальный режим 7, по умолчанию =1 (два соседних поля и одно следующее при неудаче);
- "PP" — пост-обработка, от 0 до 7, что делать с плохими кадрами (с гребенкой) при неуспехе подбора полей, имеется в виду возможность попиксельного деинтерлейса (0 — без обработки, 1 — искать плохие, выводить подсказку, но не обрабатывать, 2 — простой деинтерлейс смешением, 3 — простой деинтерлейс кубической интерполяцией, 4 — простой деинтерлейс с улучшенными краями (ELA), 5 — адаптивный к движению деинтерлейс смешением, 6 — адаптивный к движению деинтерлейс кубической интерполяцией, 7 — адаптивный к движению деинтерлейс с улучшенными краями (ELA), по умолчанию = 6).

В разработках "Tritical" TIVTC и TDeint мы видим сближение методов деинтерлейса по полям и по точкам.

```
#Пример скрипта 7.1
# Простой подбор полей для PAL без децимации
LoadPlugin("C:\Plugins\DGdecode.dll") # для MPEG2
LoadPlugin("C:\Plugins\tivtc.dll") # для TFM
MPEG2Source("d:\video\film.avi") # файл видео
TFM(mode=1, PP=6)
# делаем деинтерлейс подбором полей и пост-обработкой
```

Для мультфильмов рекомендуют режим PP=7 со сглаженными наклонными краями. Для NTSC необходима еще и функция децимации **TDecimate** сразу за **TFM**. Функции имеют многочисленные настроечные параметры, в том числе для прямого задания пользователем типа деинтерлейса для конкретных кадров.

Разработан плагин RestoreFps, а также сложные AVS скрипты для трудных случаев восстановления прогрессивных кадров из потока после телекино-преобразования, в том числе и частично смешенных (blended) полей, такие как Restore24 (составители "scharfis_brain", "Didee"), CDeblend, MRestore, CRestore (составитель "MOmonster"). Автор не является в этом вопросе опытным, однако рассмотреть их целесообразно. Ограничимся кратким описанием последней, не приводя саму функцию скрипта (найдете ее в архиве avs.zip). В ней сначала производится вычисление масок сравнения, а затем покадровые вычисления и нахождение наиболее походящих полей с использованием команд условной обработки ScriptClip и FrameEvaluate. Параметрами функции Crestore являются:

- mode: режим детектирования смешенных полей, от -1 до 10, по умолчанию 0.
  меньше 0 — быстрые режимы, больше 0 — высококачественные. Чем больше модуль, тем выше влияние движения, 0 — без влияния движения. Для природы хорошо от 0 до 2, для мультфильмов до 4-5
- rate: выходная частота кадров, обычно 23,976 (по умолчанию) или 25 fps, прореживания не производится
- bthresh: порог детектирования смешения. От 0 до 99 — простое использование порога
  Выше 99 — специальная обработка. Если текущий кадр имеет большую вероятность смешения, чем соседние, и фактор выше bthesh/100-1, то это — смешение (blend). По умолчанию 120
- dl: предел детектирования для mode >= 0. Не учитываются пиксели с меньшей разницей. От 0 для хороших источников до 2-4 для мультфильмов, по умолчанию 1
- clip2: необязательный вспомогательный (очищенный) клип (bobbed) для более надежного детектирования
- nomo: точка отсутствия движения. Разности смешенных пикселей вычитаются из нее, чистых пикселей прибавляются. Хорошие величины 255, 128. Для bthresh
- mthresh: для простого порога движения. При разности кадров меньше mthresh, смешение не будет детектировано. По умолчанию 0.16

умолчанию 0.16

Функция Crestore (в последнем варианте 1.0) использует функции плагина MaskTools новой версии 2.0 (разработчик "Manao") , на ее входе должно быть видео после Боб-деинтерлейса (кадры удвоенной частоты и полной высоты, из каждого поля, см. ниже). Функцию можно, например, импортировать из файла.

```
#Пример скрипта 7.2
# Восстановление прогрессивных кадров из потока после телекино
LoadPlugin("C:\Plugins\mt_masktools.dll") # для масок
LoadPlugin("C:\Plugins\DGdecode.dll") # для MPEG2
LoadPlugin("C:\Plugins\TDeint.dll") # для Боб-деинтерлейса
Import("crestore.avs") # импортируем файла скрипта с функцией Mpeg2Source("d:\video\film.d2v") # открываем файл видео
# подразумеваем, что из MPEG2 получен правильный порядок полей TDeint(mode=1) # делаем Боб-деинтерлейс
# И вызываем функцию для восстановления прогрессива Crestore(mode=0, rate=25, bthesh=120)
# попробуйте подобрать некоторые параметры
```

Другой случай — применение деинтерлейса с восстановлением смежного поля некоторой интерполяцией точек изображения, обычно с некоторой адаптацией к движению (по-разному в статичных и движущихся областях). Для изначально чересстрочного видео деинтерлейс вреден, так как уменьшается плавность движений, но иногда он нужен при обработке фильмов.

Имеются классический плагин KernelDeint (разработчик Donald Graft) и его оптимизированная версия LeakKernelDeint (разработчик "Leak"), великолепный быстрый TomsMoComp (разработчик Tom Barry) с небольшой компенсацией движения, и развивающийся в настоящее время продвинутый TDeint (разработчик "Tritical") с пониженной ступенчатостью краев объектов, анализом перекрывающихся областей и другими режимами.

```
#Пример скрипта 7.3
# Деинтерлейс
LoadPlugin("C:\Plugins\TomsMoComp.dll") # загружаем плагин
LoadPlugin("C:\Plugins\DeGrainMedian.dll") # загружаем плагин
AviSource("d:\video\film.avi") # аналоговое видео
AssumeTFF() # сообщим порядок полей — верхнее поле первое
#Info() # Уберите знак комментария для просмотра свойств
TomsMoComp(-1, 5, 0) # делаем деинтерлейс
DeGrainMedian(interlaced=false) # фильтруем прогрессивно
```

В данном примере мы используем вызов функции **TomsMoComp** с первым параметром -1, что означает (согласно документации) считать порядок полей тем, каким его считает AviSynth. Это довольно тонкий вопрос. В записанном видеофайле AVI не хранится информация о чересстрочности, и отсутствует информация, какое поле в кадре первое во времени. Известно, однако, что для AVI-файлов формата DV (с цифровых видеокамер) первое поле — всегда нижнее. Именно это значение считает AviSynth по умолчанию для любого видео, если нет явного задания порядка полей. Отметим, что в большинстве плат захвата аналогового видео (в том числе на чипах Philips) — верхнее поле является первым. В рассмотренном примере мы явно указали, что верхнее является первым (Top Field First). При неверно указанном порядке полей ряд функций будет работать неправильно. В файлах MPEG2 такая информация имеется, и она передается в AviSynth при использовании **MPEG2Source** пакета DGDecode.

В этом случае, как и во многих других, полезно использовать отладочный фильтр **Info()**, который печатает поверх кадра некоторую информацию о свойствах некоторого клипа (по умолчанию — LAST) в данном месте скрипта.

Выбор лучшего фильтра деинтерлейса неоднозначен. При обработке фильмов PAL рекомендую попробовать **TDeint**, который имеет большое число настроечных параметров, в том числе и режим анализа перекрывающихся областей, адаптивного перехода к автоматическому подбору наиболее подходящих соседних полей (параметр tryWeave), есть в нем и режим сглаживания ступенек (параметр mode=3). Но начать можно с настроек по умолчанию. Преимуществом TomsMoComp является компенсация движения (хоть и в пределах 1-2 пикселей). Хорошие результаты при высокой скорости обработки дает новый плагин деинтерлейса Yadif,

перенесенный мной в AviSynth из MPlayer (разработчик Michael Niedermayer).

Еще одним специальным видом деинтерлейса является так называемой Боб (английское Bob, покачивание) — преобразование каждого поля в полноформатный кадр, с сопутствующим увеличением частоты кадров вдвое, из 25 в 50 Гц (каждое поле, а теперь кадр, соответствует своему моменту времени). Это создает видео, очень плавно меняющееся во времени, что используется в ряде программ просмотра MPEG и телевидения. Во встроенном фильтре **Bob** промежуточные строки строятся путем бикубической интерполяции соседних строк (с некоторым смещением). В более разумных (Smart) бобберах частично адаптивно используется информация из соседних полей. Практически каждый фильтр деинтерлейса имеет и этот режим: **KernelBob**, **LeakKernelBob**, **TDeint**. Боб-деинтерлейс часто используют при пост-обработке для просмотра видео на мониторе в реальном времени, но и в Avisynth он тоже находит применение. Его используют при преобразовании телевизионных форматов с разной частотой кадров и прогрессивностью (PAL-NTSC). Есть у него некоторые преимущества и при фильтрации шумов. Так как имеем фактически прогрессивный выход с удвоенной частотой, то временной интервал меньше в два раза, чем при обработке временным фильтром интерлейсного клипа с разложением на четные и нечетные строки. Недостатком является меньшая скорость работы и некоторое вертикальное покачивание (bobbing) статичных объектов даже для разумных бобберов.

```
#Пример скрипта 7.4
# Фильтрация боб-деинтерлейсного видео
LoadPlugin("C:\Plugins\DeGrainMedian.dll") # загружаем плагин
AviSource("d:\video\film.avi") # аналоговое видео
AssumeTFF() # сообщим порядок полей — верхнее поле первое
TDeint(mode=1) # боб-деинтерлейс, полные кадры
# четные строки — исходные, нечетные — интерполяция,
# в следующем кадре наоборот
DeGrainMedian(interlaced=false) # фильтруем прогрессивно
AssumeFrameBased() # полагаем что кадр
SeparateFields() # разделим на поля
SelectEvery(4,1,2) #выделим из четырех второе и третье поле
Weave()
```

weave()

Отметим, что даже если некоторый фильтр деинтерлейса не имеет режима BOB (например, TomsMoComp), его можно реализовать, применяя поочередно к исходному и смещенному на одно поле видео, что может быть записано с помощи функции.

```
#Пример скрипта 7.5
# Определение функции TomsBob (составитель scharfis_brain)
function tomsbob(clip i)
{
interleave(i.tomsmocomp(-1, 0, 0), \
i.doubleweave().selectodd().tomsmocomp(-1, 0, 0))
assumeframebased()
getparity(i) ? assumetff() : assumebff()
}
```

Для использования этой функции в приведенном выше примере скрипта 7.4, запишите ее определение в скрипт (перед основным текстом), и используйте вызов **Tomsbob()** вместо строки **TDeint(mode=1)**.

8. Преобразование цветовых форматов

В отличие от большинства программ, ведущих обработку в трех цветовых каналах RGB (красный, зеленый, синий), AviSynth может обрабатывать видео в цветовом пространстве YUV (яркость Y и две цветоразности, цветности U и V), в которых обычно ведется вещание и хранится сжатое видео. Эти форматы хранения видео обычно сокращенные (прореженные) для цветности. В формате YUY2 цветность прорежена в два раза по горизонтали, а в формате YV12 (используемом в DVD) — и по вертикали. Желательно обходиться без преобразования цветового формата, но иногда это невозможно из-за несовпадения требуемого выходного формата с входным, или ограничений некоторого конкретного фильтра, в котором автором не реализована поддержка какого-то формата.

При подаче выходных кадров для кодировки в TMPGenc необходим формат RGB, как и при подключении в Avisynth плагинов от VirtualDub (такая возможность имеется), а в Canopus Procoder лучше подавать YUY2.

В AviSynth встроены все необходимые функции для преобразования (конвертации), с именами типа ConvertToXXX, где XXX — это требуемый формат (RGB, YUY2, YV12).

Обработка производится быстрее в сокращенных форматах (меньше объем данных), особенно для YV12, в котором данные по каждой цветовой плоскости хранится в отдельном массиве памяти (планарно). Рассмотрим пример, когда некоторый фильтр работает только в YV12 (это типично для новых плагинов), а источник типа YUY2.

```
#Пример скрипта 8.1
# Преобразование цветового формата
LoadPlugin("RemoveGrain.dll") # загрузим плагин заранее
AviSource("d:\video\film.avi") # аналоговое видео YUY2
AssumeTFF() # сообщим порядок полей — верхнее поле первое
# Преобразуем клип в цветовой формат YV12
ConvertToYV12(interlaced=true) # опция для чересстрочного
SeparateFields() # разделяем на поля
RemoveGrain(mode=17) # функция работает только в YV12
Weave() # собираем поля
```

В данном примере мы использовали для иллюстрации плагин RemoveGrain (разработчик Rainer Wittmann "Kassandro") — удалитель зерна (пространственный шумоподавитель) в одном из его многочисленных режимов.

В скрипте следует обратить внимание на то, что команда преобразования формата **ConvertToYV12** использована с параметром **interlaced=true**, это необходимо для правильного преобразования, если видео — чересстрочное. Еще следует обратить внимание, что преобразование цветового формата в скрипте производится с полнокадровым видео, до разделения видео на поля, то есть не рекомендуется преобразовывать цветовой формат разделенного на поля видео, иначе результат будет неверен (немного искажены цвета).

Еще следует иметь в виду, что AviSynth при преобразовании из RGB в YUV по умолчанию сжимает диапазон яркостей (а при обратном — расширяет). Позволю напомнить, что хоть диапазон, хранимый в байте, может быть от 0 до 255, и это нормальный диапазон для формата RGB, то для формата YUV ситуация другая. В телевидении допустимыми (рекомендуемыми) значениями являются от 16 до 235 для яркости и от 16 до 240 для цветности. Поэтому, строго говоря, яркость не должна выходить за эти безопасные пределы (ниже "уровня черного" и выше "уровня белого"), они должны быть обрезаны (параметр **coring** некоторых фильтров). Однако в компьютерах такие значения часто реализуются и нормально отображаются, как и в телевидении высокого разрешения HDTV. При необходимости можно произвести коррекцию, и использовать параметр **matrix** для выбора матрицы и диапазона преобразования.

9. Коррекция яркости и цвета

В AviSynth есть несколько способов коррекции цвета (встроенные фильтры Levels, Tweak, ColorYUV и некоторые другие). Наиболее мощным является ColorYUV (пожалуй, он перегружен числом режимов). Усиление, смещение, гамма и контраст могут быть установлена для каждого яркостного или цветового канала Y,U,V. Просто меняйте значения соответствующих параметров, обновляйте изображение (**F5** в VirtualDubMod) и наблюдайте результат. Однако часто визуального контроля недостаточно. В функции ColorYUV имеется очень полезный режим **Analyze=true** для анализа максимума и минимума каналов и вывода чисел на кадр.

```
#Пример скрипта 9
# Анализ и коррекция яркости (можно и цвета)
AviSource("film.avi")
ColorYUV(gain_y=10, off_y=0, gamma_y=0, cont_y=0) # яркости
ColorYUV(analyze=true) # анализ результата
```

Естественно, после настройки яркости строчку с анализом надо убрать или закомментировать.

Если диапазон YUV так широк, а нужен безопасный для телевидения, его можно сузить специальной командой **ColorYUV(levels="PC->TV")**, а если превышения небольшие, лучше подрегулировать усиление, смещение, гамму, или контраст перед обрезкой (ограничением). Обратите внимание, что анализ выдает не только абсолютные максимум и минимум, но и нестрогий (**loose**) максимум и минимум, за значения которых выходит очень небольшое число точек (часто случайных, краевых). Часто можно ими пожертвовать и ограничить, не корректируя остальную картинку.

Во многих DVD и HDTV источниках используется набор коэффициентов преобразования в RGB, слегка отличный от используемого в AviSynth и DivX, и для коррекции этого существует плагин ColorMatrix (разработчики Wilbert Dijkhof и др.) с одноименной функцией (которая может работать автоматически, получая подсказки о цветовой матрице от MPEG2Source через параметр hints).

Коррекцию цвета лучше делать в начале скрипта, но некоторые фильтры (смена размеров, повышение резкости) могут расширить интервал, поэтому при необходимости в конец можно добавить команду **Limiter()**.

Для "запущенных" случаев существует плагин автоматического локального усиления яркости в тенях HDRAGC (разработчик "pavico").

10. Изменение размеров, обрезка, бордюры

В начале века было популярно перекодирование DVD в формат MPEG4 (DivX и т.п.) и запись на CD. При этом уменьшали размер кадра, чтобы уменьшить объем информации, и чтобы программный проигрыватель справился с воспроизведением. Сейчас настало время, когда начали увеличивать размер кадра с DVD для воспроизведения на телевизорах высокого разрешения. Но иногда необходимо изменить размер кадра видеоклипа и по другим причинам. В Avisynth встроены несколько качественных и быстрых функций изменений размеров кадра путем интерполяции, отличающиеся числом опорных точек (порядком) интерполяции. Обычно для увеличения размеров рекомендуют использовать бикубическую или Ланкзос-интерполяцию, а для уменьшения размеров — Ланкзос для резкой или билинейную для мягкой (иногда другие). Изменение размеров в скрипте делается чаще после основной обработки (подавление шума, и т.п.). Отметим, что кроме точечной и билинейной, другие типы могут увеличить резкость, и несколько расширить диапазон интенсивностей (может понадобиться ограничение типа **Limiter**). При обработке чересстрочного видео, изменение вертикального размера необходимо производить после деинтерлейса (или после разложения видео на поля).

Иногда появляется задача записать на DVD видеоролик, снятый цифровым фотоаппаратом с разрешением не выше 640x480. Можно его записать в формате NTSC, другим решением будет привести его к формату DVD PAL-Video (пока обойдем вопрос другой частоты кадров).

```
#Пример скрипта 10.1
# Обработка с изменением размеров кадра видеоролика с фотика
DirectShowSource("d:\video\photo.mov")
# открываем файл типа QuickTime
LanczosResize(720,576) # новые размеры
```

Можно применить и еще более продвинутые методы увеличения размеров, в частности плагины EDIUpsizer, EEDI2, NNEDI (разработчик — "Tritical"), или более старый Sangnom (разработчик — "MarFD"), подавляющие ступеньки на диагональных линиях. Они могут качественно увеличить кадр по вертикали в два раза, после этого до нужного размера его надо довести базовыми функциями (LanczosResize).

В Avisynth есть быстрая функция **Reduceby2** сокращения вдвое, но она дает сдвинутую на полпиксела картинку. Еще стоит отметить полезность функции **PointResize** точечного увеличения (дублирования пикселей без интерполяции), она очень полезна для 2-4 кратного увеличения кадра (области) для его тонкого разглядывания.

Другим способом изменения размеров кадра является обрезка или добавление бордюров. Так, например, можно обрезать мусор при захвате VHS; естественно, при кодировании DVD число строк менять нельзя, и на место обрезанных строк нужно добавить бордюры. Это можно сделать и одной функцией **Letterbox**. Поскольку современные чипы аналогового захвата внутри работают с разрешением 704 пикселя в строке, целесообразно вести захват в данном разрешении, а потом для правильного аспекта добавить бордюры до 720.

#Пример скрипта 10.2

```
#Добавление бордюров и обрезка
AviSource("film.avi") # захваченное видео 704x576
AddBorders(8,0,8,0) # Добавим справа и слева по 8 черного
Crop(0,0,0,-16) # Обрежем снизу, например, 16 мусора
Addborders(0,0,0,16) # И добавим снизу 16 черного
```

Аналогичным образом можно сдвинуть изображение на целое число точек, а для более точного смещения можно использовать любые функции изменения размеров XXXResize с дополнительными параметрами типа src_left.) Некоторые предпочитают заливать не строго черным с резкой границей (которую желательно делать кратной 8 или 16), а с плавной, для этого есть схожие плагины BorderControl (разработчик Simon Walters) и FillMargins (разработчик Tom Barry).

11. Подавление шума

Под шумом видео мы понимаем случайные, не связанные между собой во времени и пространстве процессы, независимо в каждом пикселе. Помехи — это другое, они представляют несколько упорядоченные узоры, например, зерна (кластеры) из нескольких точек, полосы и т.п.

> **Примечание**. Подавление шума — пожалуй, одна из самых распространенных функций. Подавить его стремятся по двум причинам. Во-первых, чтобы улучшить визуальное качество, а во-вторых, чтобы улучшить сжимаемость при кодировании в MPEG. Однако, удаление шума — это всегда компромисс, фильтр неизбежно вместе с ним удалит и

часть деталей объекта. Мое мнение таково — если шума немного, то может лучше его не трогать? Просто добавьте битрейт, если это возможно. А если шума много, удалить его безболезненно тем более не получится. Лучше недочистить, чем перечистить и наблюдать пластилиновые фигуры и квантованные (ступенчатые) переходы.

За время существования AviSynth (с 2001 г.) разработано большое число различных фильтров, реализующих различные алгоритмы, и разными людьми за время существования составлено немалое количество простых и сложных комбинированных скриптов, в целях добиться идеала (оптимума для конкретного источника и вида шума).

Естественно, сложные алгоритмы дают малую скорость обработки, что может оказаться неприемлемо при большом объеме работы.

Из не потерявших актуальность фильтров отмечу пространственные Undot (удаляет отдельные выпадающие пиксели, разработчик Tom Barry), WNR (для старого AviSynth 2.0, разработчик "Thejam"), VagueDenoiser (разработчики "Lefungus", "Kurosu", "Fizick"), оба основаны на вейвлетах (коротких волнах), из временных фильтров — CNR2 (для цветности, разработчики "MarcFD", "Tritical"), PeachSmoother (разработчик "Lindsey Dubb"), оба с рекурсивной обработкой длинных последовательностей кадров), а из пространственно-временных фильтров, учитывающих соседние пиксели из текущего и нескольких смежных кадров — FluxSmooth (простой и эффективный для слабого шума, разработчик Ross Thomas), Deen (3D-конволюция, то есть свертка, разработчик "MarcFD"), Dust (с компенсацией движения, для старого AviSynth 2.0, разработчик "Steady"). Из новых фильтров отмечу RemoveGrain (высоко оптимизированный по скорости пакет из нескольких фильтров с множеством режимов, разработчик "Kassandro", у которого есть другие фильтры и форум), собственно функция **RemoveGrain** — пространственный фильтр, любопытен фильтр FrFun (использует фрактали, разработчик "prunedtree"), упомянем и "сладкую парочку" автора: DeGrainMedian (быстрый) и FFT3DFilter (медленный), оба они пространственно-временные (но последний — частотный). Подобен последнему и фильтр dfttest (разработчик "tritical"). Какой вам использовать? Трудно быть объективным, и уж, поскольку каждый автор считает свои детища лучшими, рекомендую попробовать мои. Однако и с другими фильтрами и скриптами можно получить хорошие результаты, в любом случае это только инструменты, которыми надо овладеть и применять умеючи.

Большинство фильтров имеют настроечные параметры, которыми, в частности, задается порог величины вариации сигнала, ниже которой он считается шумом, и ряд других тонкостей конкретного алгоритма. Видеосигнал часто имеет разное качество и величину шума для каналов яркости и цветности, например, высокий цветовой шум DV камер на темных сценах, цветные помехи и низкое цветовое разрешение VHS записей, при этом необходимо использовать более высокие значения параметров фильтрации для цветовых каналов U,V, чем для яркостного канала Y.

Простые скрипты для подавления шума приведены в примерах 6.1-6.4. Фильтр **DeGrainMedian** прост и быстр, он ограничивает выпадающие пиксели на основе похожести 13 диаметральных пар соседних точек (в текущем, предыдущем и последующем кадрах), компенсируя небольшие смещения (на один пиксель) и адаптивно переключаясь на пространственную фильтрацию при большом изменении интенсивности. В отличие от большинства других фильтров, которые не изменяют пиксель, если он отличается от соседей больше чем на порог, **DeGrainMedian** изменяет такой пиксель, но на ограниченную величину. Поэтому он может уменьшить весьма сильный шум с выпадающими пикселями. Режим работы (степень влияния центрального оригинального пикселя) регулируется параметром **"mode"**, а предел изменения — параметром **"limitY"** для яркости (Y) и **"limitUV"** для цветности (U,V).

При захвате аналогового видео часто наблюдается некоторая полосатость (помехи в виде горизонтальных линий). Предусмотрен специальный режим работы **DeGrainMedian**, активизирующийся параметром **"norow=true"**, при котором соседние пиксели в той строке не учитываются.

```
#Пример скрипта 11.1
#Фильтрация аналогового чересстрочного видео.
LoadPlugin("C:\Plugins\DeGrainMedian.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
DeGrainMedian(mode=1, interlaced=true, norow=true, limitY=3, limitUV=5) # включен режим подавления горизонтальных полос
# и больший предел для цветности
```

При недостаточной степени снижения шума можно увеличить пределы, но иногда более эффективным оказывается каскадное применение медианного фильтра (повтор той строки с **DeGrainMedian**, возможно с меньшими пределами), так как он очень быстр.

```
#Пример скрипта 11.2
#Фильтрация чересстрочного видео с более сильным подавлением шума.
LoadPlugin("C:\Plugins\DeGrainMedian.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
DeGrainMedian(mode=1,interlaced=true,norow=true,limitY=2,limitUV=4)
# каскадное (повторное) применение
DeGrainMedian(mode=1,interlaced=true,norow=true,limitY=2,limitUV=4)
```

Однако это, скорее, исключение. В среде профессионалов обычно распространено негативное отношение к скриптам верстки отдельных любителей, в которых нагромождено несколько фильтров подавления шума. Обычно это приводит к излишнему замыливанию картинки и накоплению артефактов. Ранее часто использовались последовательности из отдельного пространственного и отдельного временного фильтра, но с появлением более эффективных пространственно-временных (начиная с легендарного Convolution3D разработки Sebastien Lucas "Vlad59"), это, пожалуй, устарело.

По мнению многих, наиболее эффективным фильтром подавления шума в настоящее время является **FFT3DFilter**. В отличие от большинства других фильтров, он работает не с отдельными несколькими соседними точками, а с большими областями (блоками) в смежных кадрах, анализируя спектр их частот и подавляя слабые. Он не просто подавляет шум, но и улучшает изображение, например, может сделать линии более ровными (при сильных настройках может что-то и ухудшить из-за волновых эффектов, ringing).

```
#Пример скрипта 11.3
#Фильтрация DV чересстрочного видео
LoadPlugin("C:\Plugins\fft3dfilter.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
FFT3DFilter(sigma=1.5, plane=4, interlaced=true) #фильтруем все
```

FFT3DFilter(sigma=1.5, plane=4, interlaced=true) # фильтруем все

Параметр **"plane"** определяет, какие цветовые плоскости фильтровать (0 — только яркость; 1 — только U; 2 — только V; 3 — только обе цветности U,V; 4 — все). Их можно фильтровать с разной силой, определяемой параметром **"sigma"**.

```
#Пример скрипта 11.4
#Фильтрация DV чересстрочного видео
LoadPlugin("C:\Plugins\fft3dfilter.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
fft3dfilter(sigma=3, plane=3) # фильтруем цветность
fft3dfilter(sigma=2, plane=0) # фильтруем яркость с меньшей силой
```

Последовательное применение двух временных фильтров увеличивает временной радиус действия, что может быть существенно в случае компенсации движения (смотри ниже раздел 16). Решением в нашем случае может быть разделение цветовые компонент, их независимая фильтрация с разной силой, а затем объединение, например, с помощью хитрого метода (автор — "AI").

```
#Пример скрипта 11.5
#Фильтрация DV чересстрочного видео
LoadPlugin("C:\Plugins\fft3dfilter.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
YToUV(fft3dfilter(sigma=2.5, plane=1, interlaced=true).UToY(),\
fft3dfilter(sigma=3, plane=2, interlaced=true).VToY(),\
fft3dfilter(sigma=2, plane=0, interlaced=true))
```

Попутно вы видите, как в скрипте можно записать длинные строки с помощью символа продолжения строки **«\»**.

Если обе цветовых компоненты фильтруются одинаково, можно использовать и более простой вариант для раздельной фильтрации яркости и цветности и их объединения с использованием функций MergeChroma или MergeLuma.

```
#Пример скрипта 11.6
#Фильтрация DV чересстрочного видео
LoadPlugin("C:\Plugins\fft3dfilter.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
MergeChroma(fft3dfilter(sigma=2, plane=0, interlaced=true), \
fft3dfilter(sigma=3, plane=3, interlaced=true))
```

При высоком уровне импульсного шума полезна предварительная фильтрация с помощью медианного фильтра.

```
#Пример скрипта 11.7
#Фильтрация шумного аналогового чересстрочного видео
LoadPlugin("C:\Plugins\DeGrainMedian.dll") # загружаем плагин
LoadPlugin("C:\Plugins\fft3dfilter.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
# фильтруем грубо, горячие пикселы
DeGrainMedian(mode=1, interlaced=true, limitY=3, limitUV=5)
# фильтруем тонко, только яркость
FFT3DFilter(sigma=2, plane=0, interlaced=true)
```

Фильтр **FFT3DFilter** очень медленный ввиду большого объема вычислений, в последнем скрипте цветность им не фильтруется, что несколько ускоряет обработку. В качестве компромисса для фильтрации цветности возможно применение и других фильтров, например, **CNR2**.

Возможностью повысить скорость обработки является применение плагина FFT3DGPU (разработчик "tsp"), использующего процессор современной видеокарты (DirectX9) вместо центрального процессора. Он разработан на основе **FFT3DFilter** и в версии 6.2 поддерживал почти все, кроме чересстрочности и YUY2 (а в более новых версиях это поддерживается тоже). Интересной возможностью является их комбинация для обработки разных цветовых плоскостей, обработка может производиться параллельно (для повышения эффективности распараллеливания, медленные фильтры желательно ставить до **FFF3DGPU**).

```
#Пример скрипта 11.8
#Фильтрация DV видео совместно с процессором видеокарты
LoadPlugin("C:\Plugins\fft3dfilter.dll") # загружаем плагин
LoadPlugin("C:\Plugins\fft3dGPU.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
ConvertToYV12(interlaced=true) # преобразуем цвет для FFT3DGU
AssumeBFF() # первое поле — нижнее для DV
SeparateFields()
# разделим на поля, fft3dGPU v.6.2 не работает с чересстрочным
# выделим четные и нечетные поля и обработаем их раздельно
Even=SelectEven().FFT3DFilter(sigma=3, plane=3)\
.FFT3DGPU(sigma=2, plane=0)
Odd =SelectOdd().FFT3DFilter(sigma=3, plane=3)
.FFT3DGPU(sigma=2, plane=0)
Interleave(Even, Odd) # чередуем четные и нечетные
Weave() # соединим поля в кадры
```

Яркость мы фильтруем с помощью **FFT3DGPU**, а цветность — **FFT3DFilter** (можно и наоборот, или все фильтровать в GPU). Для краткости использована объектно-ориентированная запись (с точкой), в которой первым аргументом правой функции является результат (выходной клип) более левой.

В вышеприведенном скрипте мы преобразовали цветовой формат в YV12, так как **FFT3DGPU** версии 6.2 работал только в нем. В принципе, можно написать скрипт, в котором фильтрация цвета будет и в YUY2.

```
#Пример скрипта 11.9
#Фильтрация аналогового чересстрочного видео # совместно с процессором видеокарты
LoadPlugin("C:\Plugins\fft3dfilter.dll") # загружаем плагин
LoadPlugin("C:\Plugins\fft3dGPU.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
ConvertToYUY2 (interlaced=true) # если он еще не YUY2
AssumeTFF() # первое поле — верхнее для аналога
SeparateFields()
# разделим на поля, fft3dGPU v.6.2 не работает с чересстрочным
# выделим четные и нечетные поля и обработаем их раздельно
Even=SelectEven().FFT3DFilter(sigma=3, plane=3)
Even=Even.ConvertToYV12().FFT3DGPU(sigma=2, plane=0)\
.ConvertToYUY2().MergeChroma(Even)
# заменили цветность на результат FFT3DFilter
Odd =SelectOdd().FFT3DFilter(sigma=3, plane=3)
Odd=Odd.ConvertToYV12().FFT3DGPU(sigma=2, plane=0)\
.ConvertToYUY2().MergeChroma(Odd)
# заменили цветность на результат FFT3DFilter
Interleave(Even, Odd) # чередуем четные и нечетные
Weave() # соединим поля в кадры
```

Версия 8.2 FTT3DGPU поддерживает цветовой формат YUY2 и чересстрочность, так что скрипт значительно упрощается.

```
#Пример скрипта 11.9a
# Фильтрация аналогового чересстрочного видео
# совместно с процессором видеокарты, FFT3DGPU версия 8.2
LoadPlugin("C:\Plugins\fft3dfilter.dll") # загружаем плагин
LoadPlugin("C:\Plugins\fft3dGPU.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
FFT3DFilter(sigma=3, plane=3, interlaced=true) # фильтруем цветность
FFT3DGPU(sigma=2, plane=0, interlaced=true) # фильтруем яркость
```

В ряде случаев лучше использовать другие фильтры, отвечающие типу источнику и виду помех. В частности, при фильтрации мультфильмов полезно учесть наличие строго повторяющихся дублей кадров (обычно число разных рисунков меньше частоты кадров). Существует плагин Dup (разработчик Donald Graft с последователями), который анализирует поток, и, находя цепочку последовательных дублей, может их полностью смешивать (усреднять), и всю цепочку заменить на точные повторения, что эффективно подавляет шум практически без влияния на детали, без призраков от соседних кадров, и облегчает задачу сжатия.

```
#Пример скрипта 11.10
#Фильтрация мультфильмов с дублями кадров

LoadPlugin("C:\Plugins\fft3dfilter.dll") # загружаем плагин
LoadPlugin("C:\Plugins\Dup.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
# подразумеваем, что чересстрочности нет
Dup(threshold=3, blend=true) # фильтруем дубли смешением
```

Нужно подобрать значения порога threshold, в зависимости от уровня шума, чтобы исключить как ложные пропуски дублей, так и ненужную фильтрацию в действительности разных (но похожих) кадров. Фильтр имеет дополнительные параметры для отладки. При необходимости ему должен предшествовать деинтерлейс (подбор полей). При недостаточности обеспечиваемой фильтрации (особенно не дублированных кадров с быстрым движением) можно попробовать применить и дополнительные фильтры.

12. Сравнение действия фильтров

Практически всегда задаются вопросы, какой фильтр (например, шумоподавитель) применить, и с какими настройками. Можно последовать советам и рекомендациям, но целесообразно поверить все своим глазам. Необходим детальный анализ действия приложенных фильтров на исходное видео. С помощью специальных команд AviSynth можно подготовить видео для просмотра и отладки.

Первой возможностью является формирование кадра из расположенных рядом исходного и отфильтрованного видео, и просмотр в VirtualDub или каком-либо плеере. AviSynth имеет команду **StackHorizontal** для расположения двух или более клипов по горизонтали, и **StackVertical** для расположения их по вертикали. При большом размере кадра результат не помещается на мониторе, тогда можно использовать предварительную обрезку.

```
#Пример скрипта 12.1
# Просмотр результата фильтрации рядом с исходным
Avisource("h:\test.avi")
Crop(0,0,300,0) # обрежем справа
d=DeGrainMedian(mode=1,limitY=4)
StackHorizontal(last,d)
```

Загрузив скрипт в VirtualDubMod, подвинувшись на показательный кадр, можно менять значения параметров, например, **limitY**, и обновляя экран по нажатию **F5**, наблюдать изменения. Скажем, при настройке подавителей шума следим, чтобы при уменьшении шума не удалялись существенные детали, и не размывались контуры. Часто изменения малозаметны, в этом случае полезно поместить рядом еще и разницу между исходным и фильтрованным клипом (командой **Subtract**), еще лучше усиленную (например, командой **Levels**, параметры которой тоже можно подстроить).

```
#Пример скрипта 12.2
# Просмотр результата и разницы рядом с исходным
```

```
Avisource("h:\test.avi")
Crop(0,0,300,0) # обрежем справа
d=DeGrainMedian(mode=1,limitY=4,limitUV=0,interlaced=true)
ds=Subtract(last,d).Levels(117,1,139,0,255)
# Усиленная разница фильтрованного с исходным
StackHorizontal(last, d, ds)
```

В примере скрипта мы фильтровали только яркость Y, но можно фильтровать и анализировать также и (или) цветность. Можно и сразу сравнить действие двух шумоподавителей, добавив еще окошко, или три, размещенных снизу.

```
#Пример скрипта 12.3
# Просмотр результата и разницы рядом с исходным
LoadPlugin(" FluxSmooth.dll)
LoadPlugin("DeGrainMedian.dll")
Avisource("h:\nastya.avi")
assumetff()
separatefields()
# сожмем для недеформированного вида
lanczosresize(width/2,height)
Crop(100,0,200,288) # обрежем для уменьшения
d=DeGrainMedian(mode=1,limitY=3,limitUV=3,interlaced=false)
ds=subtract(last,d).Levels(117,1,139,0,255)
# Усиленная разница
f=FluxSmoothST(7,7) # другой фильтр
fs=Subtract(last,f).Levels(117,1,139,0,255)
# Усиленная разница другого
StackVertical(StackHorizontal(last, d, ds),\
StackHorizontal(last, f, fs))
```

Один из получившихся кадров результата приведен на рисунке.

Фильтры работают по-разному, хотя оба они пространственно-временные. Вверху — DeGrainMedian, внизу — FluxSmoothST (функция плагина FluxSmooth). Справа, на картинах разности, то есть удаляемого шума, видно, что ниже и выше движущейся вверх Настиной руки **FluxSmoothST** оставил ее следы (призраки) от соседних кадров. Средняя часть руки на границе света и тени не фильтруется (равномерно серая область на картине разности). **DeGrainMedian** (картинка вверху) сильнее фильтрует лицо и складки одежды на плече, смягчая детали.

Общий шум оба фильтра фильтруют неплохо, но **DeGrainMedian** удаляет более неоднородный шум.

Другим способом визуальной оценки является поочередный показ кадров по-разному отфильтрованного клипа. Так как при этом положение объектов не меняется, легко следить за какой-либо деталью без перевода взгляда. Используется функция **Interleave**. Чтобы не запутаться, можно нанести на отфильтрованные клипы надписи функцией **Subtitle**.

```
#Пример скрипта 12.4
# Просмотр результатов с чередованием
Avisource("h:\test.avi")
```

```
d1=DeGrainMedian(mode=1,limitY=4,limitUV=0,interlaced=true)\
.Subtitle("DeGrainMedian mode=1) # фильтрация
d1s=subtract(last,d1).Levels(117,1,139,0,255)
# Получили усиленную разницу
d2=DeGrainMedian(mode=2,limitY=4,limitUV=0,interlaced=true)\
.Subtitle("DeGrainMedian mode=2) # фильтрация
d2s=Subtract(last,d2).Levels(117,1,139,0,255) # Усиленная разница
Interleave(last, d1, d2, d1s, d2s) # чередуем
```

Можно переключать два скрипта, например, во вкладках программы AvsP. Вычитание клипов вместо цепочки **Subtract.Levels** можно сделать и другими средствами, например, функцией **YV12LUTxy** плагина MaskTools (разработчики "Kurosu", "Manao") (для данного цветового формата).

Отметим, что существуют и специальные программы для оценки и сравнения видео (**CompareAVS** и другие). Возможна организация других сравнений, например, чистого, искусственно зашумленного плагином AddGrain (разработчик Tom Barry) и очищенного клипов, либо анализ эффектов повышения резкости и т.п.

13. Повышение резкости

Повышение резкости преследует цель подчеркнуть края объектов и повысить различимость деталей, которая была утрачена при трансляции, записи, фильтрации и т.д. Повышение резкости — это обычно локальное повышение контраста. Следует отметить, что повышение резкости ухудшает сжимаемость клипа, поэтому делать это надо осторожно. Также при этом обычно усиливается и шум, поэтому правилом является применение фильтров повышения резкости после фильтров подавления шума. Чаще всего затрагивают только канал яркости, а каналы цветности не изменяют. Некоторые используют повышение резкости не при кодировании, а при декодировании (просмотре), в том числе и с использованием фильтров AviSynth. Есть несколько методов повышения резкости. В AviSynth встроен фильтр **Sharpen**, реализующий простой метод. Есть также плагины ASharp (разработчик "MarkFD"), TUnsharp (разработчик "Tritical"), реализующие известный метод использования маски нерезкости. Недостатком классических (линейных) методов является сопутствующее повышения уровня шума, а также образование ореолов рядом с линиями. Есть и нелинейные методы. Имеется фильтр MSharpen (разработчик Donald Graft), который сначала находит края объектов, и повышает контраст только вблизи них, не повышая шум на ровных поверхностях. Разработаны также плагины aWarpSharp (разработчик "MarkFD"), WarpSharp (набор функций, разработчик — анонимный японец) и WarpSharp (разработчик "Sh0dan") и реализующие другой метод — повышение резкости деформацией изображения (стягиванием), что наиболее применимо для мультфильмов (аниме). Имеется два (или три) плагина с функцией **XSharpen**, в которой повышение ограничено значением соседних пикселей.

Однако все эти методы несовершенны и портят картинку. Линейные методы образуют ореолы, нелинейные производят ступенчатые края у наклонных линий. Нет однозначного ответа на вопрос, что делать с (прореженной) цветностью (менять или нет), чтобы она пространственно соответствовала яркости. Известным (экстремальным) способом достижения наилучших результатов является увеличение изображения (сверхразрешение — **supersampling**), вплоть до четырех раз (!), применение функции повышения резкости, и затем уменьшение размеров до первоначальных.

```
#Пример скрипта 13.1
# Качественное повышение резкости со сверхразрешением
LoadPlugin("C:\Plugins\Warpsharp.dll") # загружаем плагин
LoadPlugin("C:\Plugins\DeGrainMedian.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
DeGrainMedian() # подавляем шум некоторым фильтром

ss=4 # вводим переменную - кратность увеличения размера кадра
LanczosResize(width*ss, height*ss) # увеличиваем размер кадров
XSharpen(strength=255,threshold=20) # повышаем резкость
LanczosResize(width/ ss, height/ ss) # восстанавливаем размер
```

В скрипте нужно настроить силу и порог повышения резкости по вкусу. При увеличении **4** ореолов нет, однако скорость обработки очень сильно падает. Да и шум усиливается.

В дидактических целях запишем этот скрипт, введя пользовательскую функцию **SSSharpen** (язык AviSynth допускает это), которая вызывается аналогично внутренним функциям или функциям из плагинов. Сначала должно быть определение функции со списком параметров и их типов (clip, int, float, string) и последующими фигурными скобками с последовательностью команд. Определение можно привести в том файле, или записать его в отдельный файл, например, **ssxsharpen.avs** и использовать в начале основного скрипта команду **Import("ssxsharpen.avs")**, или именовать файл с расширением **avsi**, например, **ssxsharpen.avsi**, и поместить его в каталог автозагрузки плагинов.

```
#Пример скрипта 13.2
# Качественное повышение резкости с функцией SSXSharpen
# Определение функции SSXSharpen
function SSXSharpen(clip input, int "ssw", int "ssh", int "xstren", int "xthresh") {
##
## SSXSharpen (from SharpenTools by mf)
## Повышает резкость изображения используя сверхразрешение.
##
## Использование: SSXSharpen()
##
## Параметры (и их значения по умолчанию):
ssw=Default(ssw, 4) # горизонтальный масштабный фактор, от 0
ssh=Default(ssh, 4) # вертикальный масштабный фактор, от 0
xstren=Default(xstren, 255) #степень повышения резкости, 0-255
xthresh=Default(xthresh, 255) #порог повышения резкости, 0-255
##
input.LanczosResize(input.width*ssw, input.height*ssh)
```

```
input.LanczosResize(input.width*ssw, input.height*ssh)
XSharpen(xstren, xthresh)
LanczosResize(input.width, input.height)
}
##################################################
# Основной скрипт
LoadPlugin("C:\Plugins\Warpsharp.dll") # загружаем плагины
LoadPlugin("C:\Plugins\DeGrainMedian.dll")
AviSource("d:\video\film.avi") # открываем видеофайл
DeGrainMedian() # подавляем шум некоторым фильтром
# вызываем функцию с параметрами по умолчанию.
SSXSharpen(last, ssw=4, ssh=4, xstren=255, xthresh=20)
```

Скрипт работает очень медленно. Сделаны попытки ограничить чрезмерную степень повышения резкости для резких деталей, и уменьшить образование ореолов без большого сверхразрешения.

Популярен длинный скрипт, оформленный в виде пользовательской функции LimitedSharpen (составитель Didee), копия находится в файле **LimitedSharpen.avs** в прилагаемом архиве **avs.zip**. Мы не будем его подробно разбирать, кратко отметим, что в функции находятся края объектов, размытое среднее значение соседних пикселей, делается простое повышение резкости, которое затем ограничивается. Функция использует плагин **WarpSharp** (японский) для повышения резкости (в одном из режимов), и плагин MaskTools (разработчики "Kurosu", "Manao"). Функции **MaskTools** часто используются в сложных скриптах, так как помогает на основе масок и других математических операций реализовывать разнообразные алгоритмы обработки вместо разработки плагинов на языках программирования.

```
#Пример скрипта 13.3
# LimitedSharpen для ограниченного повышения резкости
# Пусть плагины WarpSharp, MaskTools в автозагрузке,
# а скрипт LimitedSharpen.avs в текущем каталоге
Import("LimitedSharpen.avs")
LoadPlugin("C:\Plugins\DeGrainMedian.dll") # загружаем плагин
AviSource("d:\video\film.avi") # открываем видеофайл
DeGrainMedian() # подавляем шум некоторым фильтром
# функция повышения резкости с параметрами по умолчанию.
LimitedSharpen()
```

Функция **LimitedSharpen** предназначена для прогрессивного видео и имеет несколько параметров для настройки ее действия.

- "ss_x", "ss_y" — факторы промежуточного увеличения по горизонтали и вертикали (по умолчанию 1.5);
- "dest_x", "dest_y" — конечная ширина и высота;
- "Smode" — номер режима повышения резкости 1 — UnSharpMask, 2 — Sharpen, 3 — MinMaxSharpen (по умолчанию);
- "strength" — сила повышения резкости (по умолчанию 100);
- "radius" — радиус маски нерезкости для Smode=1 (по умолчанию 2),
- "Lmode" — номер режима ограничения 1 — жесткое, вместе с "overshoot" (по умолчанию), 2- мягкое;
- "wide" — false — использовать для ограничения мин. и макс. величины в области 3x3 краев объектов (по умолчанию), true — 5x5;
- "overshoot" — насколько повышение резкости может переходить мин. и мах. пределы (1);
- "soft" — true — слегка смягчить клип при нахождении мин. и макс. величин, false — нет (по умолчанию;
- "edgemode" — режим обработки краев (0 — весь кадр, 1 -только края, 2 — кроме краев);
- "special" — true — использовать специальный контрастный режим резкости для Smode=3;
- "exborder" — ширина бордюров кадра, исключенная из обработки.

Есть также более быстрая модификаций функции **LimitedSharpenFaster**, включенная в состав разрабатываемой новой альфа-версии 2.0 плагина **MaskTools** , а также скрипт SeeSaw (составитель Didee) для увеличения слабых деталей без чрезмерной резкости или ступенек на сильных деталях, со стабильными во времени результатами без дрожания.

Еще одна функция для умеренного повышения резкости включена в плагин **FFT3DFilter**. Используется фильтр высоких частот Гаусса с переменной частотой среза и некоторый специальный метод ограниченного повышения резкости: самые слабые частоты (с малыми амплитудами) не усиливаются, чтобы предотвратить увеличение шума; самые сильные частоты (с большими амплитудами) не усиливаются, чтобы предотвратить чрезмерное повышение резкости и образование ореолов. Степень повышения резкости максимальна для частот со средними амплитудами. **FFT3DFilter** дает более плавные переходы областей с разной степенью повышения резкости, чем **LimitedSharpen**. Что лучше, вопрос субъективный. Скорость обработки меньше, но при использовании **FFT3DFilter** и для подавления шума, повышение резкости почти не вносит дополнительного падения скорости.

```
#Пример скрипта 13.4
# Умеренное повышение резкости с FFT3DFilter
LoadPlugin("C:\Plugins\fft3dfilter.dll") # загружаем плагин
AviSource("d:\video\video.avi") # открываем видеофайл
# канал цветности только фильтруем
FFT3DFilter(sigma=2, plane=3, interlaced=true)
# фильтруем и повышаем резкость канала яркости
FFT3DFilter(sigma=1.5, plane=0, interlaced=true, sharpen=0.7)
```

Вы можете управлять как общей силой повышения резкости, так и границами (см. документацию к **FFT3DFilter**).

14. Подавление ореолов

Близкой задачей к ограниченному повышению резкости является подавление имеющихся ореолов, которые и происходят из предшествующего "тупого" повышения резкости, и часто имеются на VHS и некоторых других источниках.

Существует специальный плагин FixVHSOversharp (разработчик "MrTibs"), состоящий из двух функций (для левого и правого края), работающих отдельно для каждой строки, то есть только в горизонтальном направлении, ищущих области с яркостью, отличающейся от яркости соседних областей. Параметрами являются: порог изменения яркости в ореоле, ширина области детектирования, и смещение рабочего пикселя от контрольного.

```
#Пример скрипта 14.1
# Подавление ореолов VHS
LoadPlugin("FixVHSOverSharp2_5.dll") # загружаем плагин
AviSource("d:\video\VHS.avi") # видеофайл формата YUY2
FixVHSOversharpL(30,12,8) # левый край
FixVHSOversharp(30,14,10) # правый край
```

Фильтр работает по довольно простому алгоритму, предназначен для сильных ореолов, и может в ряде случаев произвести артефакты, в частности на контрастных титрах.

Делался ряд попыток составить специальные скрипты для более эффективного удаления ореолов, из которых, пожалуй, наиболее удачным (и последним), является **Dehalo_alpha** (составитель "Didee"), несмотря на статус версии "альфа". Скрипт использует плагин **MaskTools** (разработчики "Kurosu", "Manao") и плагин **Repair** из пакета **RemoveGrain** (разработчик "Kassandro").

```
# Пример скрипта 14.2
# Определение функции DeHalo_alpha (составитель Didee)
function DeHalo_alpha(clip clp, float "rx", float "ry", float "darkstr", float "brightstr", float "lowsens", float "highsens", float "ss")
{
rx = default( rx, 2.0 )
ry = default( ry, 2.0 )
darkstr = default( darkstr, 1.0 )
brightstr = default( brightstr, 1.0 )
lowsens = default( lowsens, 50 )
highsens = default( highsens, 50 )
ss = default( ss, 1.5 )
#
LOS = string(lowsens)
HIS = string(highsens/100.0)
DRK = string(darkstr)
BRT = string(brightstr)
ox = clp.width()
oy = clp.height()
uv = 1
uv2 = (uv==3) ? 3 : 2
#
halos = clp.bicubicresize(m4(ox/rx),m4(oy/ry))\
.bicubicresize(ox,oy,1,0)
are = yv12lutxy(clp.expand(U=uv,V=uv), \
clp.inpand(U=uv,V=uv),"x y -","x y -","x y -",U=uv,V=uv)
ugly = yv12lutxy(halos.expand(U=uv,V=uv),\
halos.inpand(U=uv,V=uv),"x y -","x y -","x y -",U=uv,V=uv)
so = yv12lutxy( ugly, are, \
"y x — y 0.001 + / 255 * "+LOS+" — y 256 + 512 / "+HIS+" + *")
lets = maskedmerge(halos,clp,so,U=uv,V=uv)
remove = (ss==1.0) ? clp.repair(lets,1,0)
\ : clp.lanczosresize(m4(ox*ss),m4(oy*ss))
\ .logic(lets.expand(U=uv,V=uv)
\ .bicubicresize(m4(ox*ss),m4(oy*ss)),"min",U=uv2,V=uv2)
\ .logic(lets.inpand(U=uv,V=uv)
\ .bicubicresize(m4(ox*ss),m4(oy*ss)),"max",U=uv2,V=uv2)
\ .lanczosresize(ox,oy)
them = yv12lutxy(clp,remove,\
"x y < x x y — "+DRK+" * — x x y — "+BRT+" * — ?",U=2,V=2)
#
return( them )
}
# вспомогательная функция
function m4(float x) {return(x<16?16:int(round(x/4.0)*4))}
```

Функция находит области ореолов путем сравнения результата расширения и сужения масок цветовых каналов в оригинальном и сглаженном путем изменения размера кадров, и так далее... Новичкам необходимо знать, что в этом и других сложных скриптах часто используют полезную функцию **YV12LUTxy** из плагина **MaskTools**, которая производит попиксельные вычисления результата, исходя из значений пикселей в кадрах двух клипов-аргументов (x и y), при этом формула вычисления задается строкой в обратной записи (например, "x y -" означает из пикселей первого клипа X вычесть значения второго Y).

Можно иметь функцию **DeHalo_alpha** в отдельном файле **DeHalo_alpha.avs**, и затем импортировать в основной скрипт. Функция имеет следующие параметры:

- "rx" , "ry" — радиусы по горизонтали и вертикали для удаления ореолов (от 1 до 3, по умолчанию 2);
- "darkstr", "brightstr" — факторы степени для темных и светлых ореолов (от 0 до 1.0 нестрого, по умолчанию 1.0);
- "lowsens" — чувствительность к слабым изменениям, полностью принимаемым (от 0 до 100, по умолчанию 50);
- "highsens" -чувствительность к сильным изменениям, полностью отвергаемым (от 0 до 100, по умолчанию 50);
- "ss" — фактор промежуточного сверхразрешения для избегания ступенек (по умолчанию 1.5)

Функция **DeHalo_alpha** разрабатывалась для прогрессивного видео цветового формата YV12. Чересстрочное видео необходимо

Функция **DeHalo_alpha** разработана для прогрессивного видео цветового формата YV12. Чересстрочное видео необходимо предварительно разделить на поля, и целесообразно уменьшить вертикальный радиус ореола вдвое.

```
# Пример скрипта 14.4
# Удаление ореолов с функцией Dehalo_alpha
LoadPlugin("MaskTools.dll")
LoadPlugin("RepairS.dll")
Import("Dehalo_alpha.avs") # импортируем функцию в скрипт
AviSource("d:\video\VHS.avi") # видеофайл (VHS) формата YUY2
ConvertToYV12(interlaced=true)
SeparateFields() # делим на поля
Dehalo_alpha(rx=2.0, ry=1.0) # уменьшаем вертикальный радиус,
# так как поля половинной высоты
Weave()
```

По всей видимости, подавление шума лучше делать до удаления ореолов, для более четкого их нахождения. Применение функции подавления ореолов может смазать некоторые детали изображения. Попробуйте подобрать оптимальные параметры.

Отметим, что возможны и другие решения проблемы, например, в частотной области с плагином **FFT3DFilter**. Пока они не отработаны, но как предварительный вариант попробуйте вместо **DeHalo_alpha** следующую команду, использующую повышение резкости с обратным знаком для понижения высоких частот с большой амплитудой:
fft3dfilter(bt=-1,sharpen=-1.5,smin=200,smax=90000,scutoff=0.45,interlaced=true).

Снижение резкости и ореолов более плавное, чем в **Dehalo_alpha**. В новых версиях **FFT3DFilter** появилась специальная опция "dehalo", что может позволить чистить шумы, повышать резкость и подавлять ореолы одной командой.

15. Подавление радужных полос

Из многообразия дефектов видео рассмотрим встречающиеся при обработке аналоговых сигналов радужные полосы, наблюдаемые в местах с высоким контрастом яркости, часто на титрах. Дефекты эти свойственны композитному (смешанному) сигналу, из-за перекрестного влияния яркости и цветности, если не используется качественный аппаратный (так называемый гребенчатый) фильтр для подавления этих помех. Есть плагины GuavaComb (разработчик "Lindsey Dubb"), TComb (разработчик "tritical"), BiFrost (разработчик Fredrik Mellbin) для устранения подобных помех методом усреднения во времени соседних кадров (главным образом для NTSC, и только для статичных сюжетов). Другим методом является пространственное сглаживание подобных участков. В наиболее эффективном алгоритме **Rainbow Killer** («убийца радуг») сначала ищутся резкие края объектов в яркостном канале, а затем производится размытие каналов цветности вблизи найденных краев. Алгоритм реализован в скрипте-функции DeRainbow (составитель "sh0dan"). Используются плагины Msharpen (разработчик Donald Graft) для построение маски краев, MipSmooth (разработчик "sh0dan") для размытия, и **MaskTools** (разработчики "Kurosu", "Manao") для объединения. Отметим, что функция **DeRainbow** подразумевает прогрессивный входной клип формата YV12, для цветового формата YUY2 есть аналогичная функция DeRainbowYUY2. Для чересстрочного видео используйте разбивку на поля. Подавление шума (и повышение резкости) лучше делать после удаления радуги.

```
# Пример скрипта 15.1
# Функция DeRainbow (от sh0dan) для удаления радужных полос
# Использует плагины Msharpen, MipSmooth, MaskTools
function DeRainbow(clip org, int "thresh")
{
assert(org.isYV12(),"DeRainbow() requires YV12 input!")
# проверка формата с сообщением
thresh = default(thresh, 10)
# значения параметра порога по умолчанию =10
org_u = UtoY(org) # из цветности U в яркостный канал
org_v = VtoY(org) # из цветности V в яркостный канал
msharpen(org, threshold = thresh, mask=true)
# маска резких краев
bilinearresize(last.width/2, last.height/2)
# маска половинной высоты
greyscale() # в градации серого
uv = blur(1.5).levels(0,2.0,255,0,255, coring=false) \
.blur(1.5).blur(1.5).levels(50,2.0,255,0,255, coring=false)
# расширяем в маске найденные края
filtered_u = org_u.mipsmooth(spatial=255, temporal=255,\
scenechange=3 ,show=false, method="strong", scalefactor=0.5)
# размываем бывший канал цветности U
filtered_v = org_v.mipsmooth(spatial=255, temporal=255,\
scenechange=3 ,show=false, method="strong", scalefactor=0.5)
# размываем бывший канал цветности V
u_final = MaskedMerge(org_u, filtered_u, uv)
# заменяем U на размытый там где маска
v_final = MaskedMerge(org_v, filtered_v, uv)
# заменяем V на размытый там где маска
return YtoUV(u_final, v_final, org)
# заменяем U, V оригинала на фильтрованные
}
#######################################################
# Функция для удаления радужных полос для формата YUY2
function DeRainbowYUY2(clip org, int "thresh")
{
assert(org.isyuy2(),"DeRainbowYUY2() requires YUY2 input!")
```

```
thresh = default(thresh, 10)
org_yv12 = org.converttoyv12()
# преобразуем в YV12 для MaskTools и скорости
org_u = utoy(org).converttoyv12()
org_v = vtoy(org).converttoyv12()
msharpen(org_yv12, threshold = thresh, mask=true)

bilinearresize(last.width/2, last.height)
greyscale()
uv = blur(1.5).levels(0,2.0,255,0,255, coring=false)\
.blur(1.5).blur(1.5).levels(50,2.0,255,0,255, coring=false)
filtered_u = org_u.mipsmooth(spatial=255, temporal=255,\
scenechange=3,show=false,method="superstrong",scalefactor=0.5)
filtered_v = org_v.mipsmooth(spatial=255, temporal=255, \
scenechange=3,show=false,method="superstrong",scalefactor=0.5)
u_final = MaskedMerge(org_u, filtered_u, uv).converttoyuy2()
v_final = MaskedMerge(org_v, filtered_v, uv).converttoyuy2()
return ytouv(u_final, v_final, org)
}
#######################################################
LoadPlugin("C:\Plugins\DeGrainMedian.dll") # загружаем плагин
AviSource("d:\video\video.avi")
# видеофайл, пусть формат YUY2 чересстрочный
A=Trim(0,1234) # пусть первая часть клипа без радуги
B=Trim(1235,4567) #пусть вторая часть клипа, имеется радуга
C=Trim(4568,0) # пусть третья часть клипа до конца — без радуги
B=SeparateFields(B) # разделяем на поля
B=DeRainbowYUY2(B,10) # подавляем радугу в клипе B
B=Weave(B) # соединяем поля
A ++ B ++ C # объединяем клипы (с синхронизацией)
DeGrainMedian(limity=2,interlaced=true) # подавляем шум
SeparateFields() # разделяем на поля
LimitedSharpen() # повышаем резкость
Weave() # соединяем поля
```

16. Компенсация движения

Оценка движения и его компенсация — сложный способ повышения качества обработки при понижении шума, деинтерлейсе, изменении частоты (а в кодировщиках в MPEG эта информация используется для повышения степени сжатия).

Рассмотрим случай проводки камеры (панорамирования) при съемке и ее влияние на динамическое понижение шума. При этом все объекты в соседних кадрах не совпадают по расположению, поэтому глупый алгоритм фильтрации (усреднения) даст призраки по всему кадру, а умный алгоритм будет вынужден уменьшить степень фильтрации также почти по всему кадру. Если мы сдвинули бы предыдущий кадр на величину перемещения камеры, то почти все объекты совпали бы, и эффективность понижения шума значительно увеличилась.

Такое глобальное движение (смещение, зум, вращение) позволяет оценить и компенсировать плагин DePan (разработчик "Fizick"), который включает функцию-сервер оценки движения **DepanEstimate**, производящую клип в специальном формате со вставленными в первые строки данными о движении, и несколько функций-клиентов, использующих эти данные для компенсации.

```
#Пример скрипта 16.1
# Шумоподавление с компенсацией глобального движения
LoadPlugin("Depan.dll")
LoadPlugin("DeGrainMedian.dll")
AviSource("test.avi")
AssumeTFF() # для аналогового чересстрочного видео
SeparateFields()
M=DepanEstimate(trust=3) # спецклип M с данными по движению
DepanInterleave(data=M, prev=1, next=1) # утроенное число кадров
DeGrainMedian() # некоторый шумопонижающий фильтр
Selectevery(3,1) # выделим из каждых трех один кадр

Weave()
```

В скрипте использована функция **DePanInterleave**, которая для каждого текущего кадра подает на выход тройку похожих кадров: скомпенсированный по движению вперед предыдущий, текущий и скомпенсированный по движению назад последующий. Совместно с последующей прореживающей командой SelectEvery, они образуют как бы скобки (обкладки), внутри которых пишем временной (пространственно-временной) фильтр. Отметим, что такая обработка не имеет смысла с чисто пространственными фильтрами, для них компенсация не нужна, их можно указать до утроения. Еще обратим внимание, что для некоторых временных фильтров, с большим временным радиусом (больше одного кадра), нужно соответственно увеличить число компенсированных кадров в группе (параметрами **prev, next**), и соответственно изменить параметры SelectEvery. В частности, при последовательном применении двух временных фильтров их радиусы складываются.

```
#Пример скрипта 16.2
# Шумоподавление двумя фильтрами
# с компенсацией глобального движения
LoadPlugin("Depan.dll")
LoadPlugin("DeGrainMedian.dll")
LoadPlugin("FFT3DFilter.dll")
AviSource("test.avi")
```

```
AviSource("test.avi")
AssumeTFF() # для аналогового чересстрочного видео
SeparateFields()
M = DepanEstimate(trust=3)
# получен спецклип M с данными по движению
DepanInterleave(data=M, prev=2, next=2)
# упятеренное число кадров, временной радиус фильтров равен 2
DeGrainMedian() # некоторый временной фильтр
FFT3DFilter() # второй временной фильтр
Selectevery(5,2) # выделим из каждых пяти второй кадр с 0
Weave()
```

Способ обработки чересстрочного видео неоднозначен. Можно, например, выделить и отдельно анализировать и компенсировать четные и нечетные поля, но при этом увеличится временной интервал, то есть, уменьшится схожесть изображений. Пожалуй, наиболее правильным является использования разумного Боб-деинтерлейса, создающего полноформатный выход с соудвоением частоты и, хотя при этом уменьшится скорость обработки. Кстати, функции **DePanEstimate** и **DepanInterleve** имеют ряд параметров для тонкой настройки, в частности, скорость оценки движения можно повысить установкой параметра **fftw** (с использованием и внешней библиотеки FFTW3.DLL).

```
#Пример скрипта 16.3
# Шумоподавление с компенсацией глобального движения
# и Bob-деинтерлейсом
LoadPlugin("Depan.dll")
LoadPlugin("LeakKernelDeint.dll")
LoadPlugin("DeGrainMedian.dll")
AviSource("test.avi") # пусть аналоговое чересстрочное
LeakKernelBob(order=1) # для TFF
AssumeTFF() # LeakKernelBob не сохраняет доминантность полей
M=DepanEstimate(trust=3, fftw=true)
# спецклип M с данными по движению
DepanInterleave(data=M,prev=1,next=1)#утроенное число кадров
DeGrainMedian() # некоторый шумопонижающий фильтр
Selectevery(3,1) # выделим из каждых трех один кадр
SeparateFields()
selectevery(4,0,3)#выделяем исходные не интерполированные поля
weave()
```

В приведенном скрипте при отсутствии шумоподавителя исходный клип передается на выход неизменным, а два уровня "скобок" (внешний **Bob** и внутренний **Depan**) стремятся сделать соседей фильтруемого кадра (поля) как можно более на него похожими.

Еще одним использованием глобального движения является такая его частичная компенсация, при которой происходит снижение тряски кадра, как в аппаратных стабилизаторах видеокамер. Один из простых алгоритмов стабилизации реализован в функции **DepanStabilize**.

```
#Пример скрипта 16.4
# Стабилизация тряски
LoadPlugin("Depan.dll")
LoadPlugin("DeGrainMedian.dll")
AviSource("test.avi")
AssumeTFF() # для аналогового чересстрочного видео
SeparateFields()
M=DepanEstimate(trust=3) # спецклип M с данными по движению
DepanStabilize(data=M, cutoff=1, mirror=15) # стабилизация
Weave()
DeGrainMedian(interlaced=true) # фильтрация после стабилизации
```

Некоторую стабилизацию можно реализовать также с помощью плагина Motion (разработчик "mg262"), возможно также использования известного плагина DeShaker (разработчик Gunnar Thalin) для VirtualDub, с которым **DePan** имеет общий формат лог-файлов.

При помощи частичной компенсации глобального движения также можно построить кадры, соответствующие промежуточным моментам времени, что можно использовать, например, для повышения частоты кадров старых любительских узкопленочных фильмов, съемки панорам становятся более плавными.

Однако компенсация глобального движения эффективна не всегда, а только если движение действительно в основном глобальное. Если движение объектов существенно разнится, то необходим анализ и компенсация локального движения, что реализовано в плагинах MVTools (разработчики "Manao", "Fizick") и Motion (разработчик "mg262"), которые ведут анализ поблочно. Плагин **Motion** высоко оптимизирован по скорости, но оценка движения более точна у плагина **MVTools**, который и будет описан в данной статье.

Использование компенсации локального движения для целей понижения шума аналогично рассмотренному использованию глобального движения. Для каждого блока в текущем кадре находится похожий на него блок в предыдущем, компенсацией пиксели ставятся на соответствующие места, и производится фильтрация. Впервые для AviSynth это было реализовано в известном плагине **Dust** (разработчик "Steady"), который был в свое время самым лучшим и медленным шумоподавителем. В плагине **MVTools** имеется встроенная функция понижения шума **MVDenoise** а также новая усовершенствованная **MVDegrain2**. Более универсальным (это не означает оптимальным) решением является независимая компенсация движения и использование с ним произвольного временного или пространственно-временного шумоподавителя.

```
#Пример скрипта 16.5
# Шумоподавление с компенсацией локального движения
LoadPlugin("MVTools.dll")
LoadPlugin("LeakKernelDeint.dll")
```

```
LoadPlugin("LeakKernelDeint.dll")
LoadPlugin("DeGrainMedian.dll")
AviSource("test.avi") # пусть аналоговое чересстрочное
LeakKernelBob(order=1) # Bob-деинтерлейс для TFF
AssumeTFF() # LeakKernelBob не сохраняет доминантность полей
Vf=MVAnalyse(isb=false, truemotion=true)
# спецклип Vf с векторами движения вперед
Vb=MVAnalyse(isb=true, truemotion=true)
# спецклип Vb с векторами движения назад
f=MVFlow(vectors=vf)
b= MVFlow(vectors=vb)
Interleave(f, last, b)
DeGrainMedian() # некоторый временной шумопонижающий фильтр

Selectevery(3,1) # выделим из каждых трех один кадр
SeparateFields()
selectevery(4,0,3) # выделяем исходные не интерполированные поля
weave()
```

Плагин **MVTools** имеет в составе ряд функций, в частности для создания маски движения **MVMask**, для изменения частоты кадров **MVFlowFps** и др. Описание и простые базовые примеры их использования хорошо описаны в документации. Рядом пользователей составлены и более сложные скрипты с использованием **MVTools**, в частности с большим числом компенсированных кадров для сильного понижения шума.

Стоит обратить внимание и на попытки реализовать компенсированный по движению деинтерлейс, из которых одной из наиболее удачных признана скриптовая функция **MVBob** (составитель "scharfis_brain"), приведенная в файле **mvbob.avs** (вместе со вспомогательными функциями) в прилагаемом архиве **avs.zip**.

Она использует упоминавшиеся выше плагины **MVTools**, **MaskTools**, **TomsMoComp**, **LeakKernelDeint**, **Eedi2**, **Undot** и **RemoveGrain**, и имеет следующие именованные параметры:

"blksize" — размер блоков (по умолчанию = 8)
"pel" — точность (по умолчанию =2, субпиксельная)
"th" — порог для коррекции ошибочной компенсации (по умолчанию = 8)
"ths" — порог для коррекции статичных областей (по умолчанию = 3)
"bobth" — порог деинтерлейса для клипа анализа движения (по умолчанию = 6)
"quick" — предварительный боб — быстрый (по умолчанию = false, то есть медленный Eedi2, иначе – Tomsbob)

MVBob по назначению реализует Боб-деинтерлейс, и может быть использована для организации "обычного" деинтерлейса просто выбором, например, только четных кадров (или только нечетных, или попеременно) из результата.

```
#Пример скрипта 16.6
# Качественный деинтерлейс с компенсацией движения
LoadPlugin("MVTools.dll")
LoadPlugin("MaskTools.dll")
LoadPlugin("TomsMoComp.dll")
LoadPlugin("LeakKernelDeint.dll")
LoadPlugin("EEDI2.dll")
LoadPlugin("Undot.dll")
LoadPlugin("RemoveGrainS.dll")
# В оригинальном файла mvbob.avs был вызов всех плагинов,
# мы их там закомментировали
Import("mvbob.avs")
AviSource("test.avi") # пусть аналоговое чересстрочное
AssumeTFF() # Измените для DV на AssumeBFF
ConvertToYV12(interlaced=true) # пока поддерживается YV12
MVBob() # Боб-деинтерлейс с параметрам по-умолчанию
SelectEven() # выбрали четные кадры (ведущее поле четное)
```

Другой известной аналогичной функцией является скрипт MCBob (составитель — "Didee"), использующий поблочную компенсацию MVCompensate и поэтому несколько менее медленный.

Отметим, что алгоритмы анализа и компенсация движения не совершенны (плагины и скрипты находятся в процессе разработки), и возможны некоторые артефакты. Ряд защитных мер их по предотвращению уменьшает и без того невысокую скорость обработки.

Самой сложной проблемой является не компенсация, а **интерполяция** движения, то есть восстановление (реконструкция) местоположения и формы движущихся объектов изображения в промежуточные времена между кадрами, например, для изменения частоты кадров, преобразования PAL-NTSC и т.п. (темы заслуживают особого рассмотрения и обсуждаются). Преобразование частоты проводится с прогрессивным видео, а для чересстрочного предварительно применяется деинтерлейс (типа Боб для видеокамеры или подбор полей и прореживания для фильмов), затем преобразование частоты с последующим выделением нужных полей и преобразованием видео в чересстрочное (при необходимости). Используемые в функции **MVFlowFps** (и в функциях-скриптах плагина Motion типа **MotionProtectedFps**) методы могут дать для многих видео-отрезков достаточно неплохие результаты, на уровне лучших коммерческих пакетов.

```
#Пример скрипта 16.7
# Удвоение частоты видео с компенсацией движения
LoadPlugin("MVTools.dll")
# предполагаем прогрессивный источник
source = AviSource("film.avi")
# обратные вектора движения
```

```
# обратные вектора движения
backward_vec = source.MVAnalyse(overlap=4, isb = true, idx=1)
# используем перекрытие блоков overlap для качества
# прямые вектора движения
forward_vec = source.MVAnalyse(overlap=4, isb = false, idx=1)
# удвоим частоту, удваивая ее числитель и не меняя знаменатель
source.MVFlowFps(backward_vec, forward_vec,\
num=2*FramerateNumerator(source), \
den=FramerateDenominator(source), idx=1)
```

Но в ряде случаев потребуется кропотливая работа по подбору параметров и составлению скрипта для борьбы с артефактами, с медленной обработкой и без гарантии получения приемлемых результатов для всех кадров ввиду ряда причин (заслонения объектов, наличия повторяющихся схожих элементов, вспышек, очень быстрых движений, изменения формы, ограничений и недостатков используемого метода). В таком случае можно использовать и более простые методы изменения частоты, аналогичные используемым подавляющим большинством монтажных программ и кодировщиков (или просто положиться на них). В частности, встроенная в AviSynth функция **ConvertFps** работает быстро и использует смешение (blend) соседних кадров с весом, пропорциональным близости кадра к рассчитываемому моменту времени. Рекомендуется, если это, возможно, не терять оригинальных кадров (полей) при преобразовании, то есть увеличивать частоту кратно, и, например, попробовать преобразовать фотоаппаратное 15 Гц видео в стандартный формат NTSC 30 Гц (точнее 29,97), а не в PAL 25 Гц. Есть и функция **ChangeFps**, реализующая метод дублирования кадров, что позволяет избежать их изменения, но за счет введения прерывистости движения. Стоит отметить и наличие функции **AssumeFps**, которая не меняет последовательность и число кадров, а просто заставляет AviSynth полагать, что частота равна новому заданному значению (удобно для малых изменений частоты захваченного видео типа 24,995, но может потребоваться подгонка звука). Может быть организовано и изменение частоты путем преобразования из прогрессивного в чересстрочное видео с повтором некоторых полей по определенному шаблону с использованием **DoubleWeave**, **SelectEvery** и **Weave** (фактически это промышленный стандартный метод Pulldown, используемый при записи фильмов в формате NTSC, и, кстати, может быть реализован прямой модификацией готового MPEG2 файла программой DGPullown без участия AviSynth).

Вы можете сами выбрать свой "яд" (плавность с возможными артефактами, смешение кадров или прерывистость).

17. Обработка звука

Наряду с обработкой изображений, AviSynth имеет ряд базовых возможностей и для обработки звука. Естественно, при вводе видеофайла считывается и звук (в некоторых случаях, например, для MPEG2, может потребоваться дополнительная команда считывания звука). Точнее, он, как и видео, распаковывается, поэтому на выходе из AviSynth формат звука — всегда несжатый PCM. Следовательно, если вы, например, хотите перекодировать MPEG2 видео в MPEG2 и не хотите перекодировать звук, его надо заранее демультиплексировать (отделить от видео), например, в программе DGIndex пакета DGDecode, и после очистки и кодирования видео — мультиплексировать (при сохранении длительности). Звук привязан к видео, при нарезке видео отрезаются и соответствующие участки звука. Выше в скриптах мы использовали при объединении клипов оператор двойного плюса **++**. Это эквивалент функции **AlignedSplice**, которая, в отличие от одиночного плюса (**UnAlignedSplice**), подрезает первую звуковую дорожку или вставляет тишину при необходимости, чтобы вторая звуковая дорожка оставалась синхронизированной с видео. Звук можно считывать и отдельно от видео с помощью команды **WAVSource**. Форматы аудио, встречающиеся на DVD (AC3, DTS, MPA , LPCM) можно импортировать с помощью плагина NicAudio (разработчик "Nic"). Для чтения других форматов можно попробовать BassAudio (разработчик "dimzon") и FFMpegSource. В AviSynth имеются команды усиления **Amplify**, нормализации **Normalize**,

задержки **DelayAudio**, эквалайзер **SuperEQ**, простого изменения частоты дискретизации **ResampleAudio** и высококачественного изменения частоты дискретизации **SSRC**, и даже изменения скорости воспроизведения звуков без изменения высоты и изменения высоты звуков без изменения длительности звука **TimeStretch**. Для многоканального звука есть выбор каналов **GetChannel**, и их добавление **MergeChannels** или смешивание **MixAudio**.

Рассмотрим в качестве примера задачу обработки любительского видео, когда надо к определенному отрезку подмешать для озвучивания некоторый музыкальный фон.

```
#Пример скрипта 17.1
# Подмешивание звука к видео
video = AviSource("test.avi") # источник видео
# видео содержит одноканальный моно звук с частотой 48 кГц
# приведем его в двухканальный (стерео) дублированием
video = MergeChannels(video, video)
# audio = WavSource("music.wav") # источник музыки
# предположим аудио содержит звук с частотой 44.1 кГц (CD)
# приведем его к частоте дискретизации 48 кГц
audio = audio.ResampleAudio(48000)
video # делаем Last равным video
trim(0,1000) ++ trim(1001,5000).MixAudio(audio, 0.5) \
++ trim(5001,0)
# к средней части видео подмешали звук музыки
```

Плавного затухания или перетекания видео вместе со звуком можно добиться с использованием функции **FadeIn** и подобных.

Подстановку (вставку) звука из одного клипа в другой, точнее комбинацию видео и аудио из двух клипов, можно произвести функцией **AudioDub**.

```
#Пример скрипта 17.2
# Комбинирование (соединеие) видео с аудио для DVD
# LoadPlugin("C:\dgdecode.dll") # плагин для чтения MPEG2 видео
# LoadPlugin("C:\nicaudio.dll") # плагин для чтения MPEG2 аудио
# Откроем DVD видео, указав файл индекса D2V
MPEG2Source(d2v="mydvd.d2v") # читаем видео в подразумеваемый Last
audio=NicAC3Source("mydvd.ac3") # читаем звук в переменную audio
```

или NicMPASource, NicLPCMSource, NicDTSSource
AudioDub(last,audio) # видео из клипа last и звук из клипа audio
результат помещается в новый клип Last

Экспорт аудио можно организовать с помощью функции (плагина) SoundOut (разработчик Klaus Post "sh0dan"). Как и в случае с видео, существуют некоторые программы-утилиты, использующие AviSynth, в частности BeHappy (разработчик "dimzon") — основанный на AviSynth инструмент для транскодирования аудио.
Есть утилита командной строки Avs2wav (разработчик "Kassandro") для чтения AVS скриптов и генерации WAV файлов или WAV аудио потоков (на вход некоторых программ-кодировщиков).

Хочется закончить статью на мажорной ноте (имеется команда **Tone**, проиграйте файл Authors.avs в подкаталоге Examples), но если ничего не нравится и хочется тишины, то есть команда полного удаления звука **KillAudio**. Кстати, последним достижением новейшей версии AviSynth 2.5.7 явилась команда **KillVideo**, позволяющая, наконец, довести обработку видео (в том числе удаление помех) до абсолютного идеала (в более старых версиях имелась лишь команда **BlankClip**, генерирующая черный экран и молчание.

А еще что есть?

Кроме рассмотренных в настоящей статье, AviSynth имеет много других возможностей. Он неисчерпаем, как и атом (и любой язык, но надо учиться...). Имеются, в частности, функции условного выполнения, также есть плагины для выполнения других специфических задач (например, реставрации старого видео, создания эффектов переходов, и т.д.). Советую ознакомиться с документацией, в том числе посмотреть "продвинутые" темы и описания плагинов, даже Вами не используемых — найдете много познавательного от квалифицированных авторов! Много полезной (хотя иногда противоречивой) информации можно найти в специализированных форумах. Официальный сайт это достаточно интересное явление, хоть и не всегда понятное в навигации — Richard Berg сделал все его Wiki-страницы с возможностью редактирования посетителями, и вы можете, например, добавить страничку с описанием своего скрипта. Программистам с начальными знаниями C++ (или с некоторыми ограничениями Pascal, Delphi) не составит труда освоить AviSynth FilterSDK и написать новый плагин для реализации своих собственных идей (проще чем для VirtualDub). Хоть формального полного описания SDK не существует, много информации есть на официальном сайте, дистрибутиве, а также в открытых исходниках имеющихся плагинов. Остальные, пробуйте силы в скриптописании!

Основные ссылки по теме:

- официальный сайт AviSynth: http://www.avisynth.org;
- страница проекта AviSynth и загрузки дистрибутива: http://sourceforge.net/projects/avisynth2;
- коллекция плагинов: http://www.avisynth.org/warpenterprises (внимание - там не всегда последние версии, рекомендую добраться до домашних страниц авторов, используйте форумы и поиск);
- русские ресурсы AviSynth и плагины разработки Fizick: http://avisynth.org.ru;
- русскоязычная ветка форума "Экстремальный Ависинт": http://forum.ixbt.com/topic.cgi?id=29:9331;
- англоязычный форум по использованию AviSynth: http://forum.doom9.org/forumdisplay.php?f=67;
- англоязычный форум по разработке AviSynth: http://forum.doom9.org/forumdisplay.php?f=69.

Рассмотренные в статье большие скрипты приведены в прилагаемом файле **avs.zip**, автор не является составителем данных скриптов.

Не стесняйтесь читать, пробовать, спрашивать, писать о достижениях и давать советы на форумах!

Автор благодарит **AI** за обсуждение черновика статьи.

Спасибо многим другим участникам форума iXBT.com за последующие комментарии и поправки.

[Все статьи в разделе «Цифровое Видео»]

1 мая 2006 г. Александр Балахнин

Обсудить в конференции (комментариев: 104)

Другие обсуждения в конференции:

- [21:36] XviD4PSP 6.0 - правильный медиа кодер (2452 сообщений)
- [21:25] Panasonic HC-V700 , модель 2012 года - у кого есть вопросы - велк... (1195 сообщений)
- [20:53] AVCHD 50\60p Panasonic SD\TM\HS\X 600/700/707/650/750/800/900 (чи... (3957 сообщений)
- [20:45] Экстрим, экшн HD камеры. Сравнение, обсуждение, выбор. (598 сообщений)
- [20:24] Выбор надёжных miniDV кассет! (503 сообщений)
- [19:24] Пара вопросов освоившим покадровый захват с 8мм кинопленки. (част... (896 сообщений)
- [18:50] Adobe Premiere Pro CS5: обсуждаем проблемы, глюки. (1258 сообщений)

Copyright © 1997—2012, iXBT.com.

Создание сайта — студия Explosion

Код для блога бета

- Код
- Предпросмотр

Выделите HTML-код в поле, скопируйте его в буфер и вставьте в свой блог.