Список использованных источников

1. Князев, В. Экологическая экономика: принципы и стратегии устойчивого развития. 2019
2. Белова, Т. Экологические инновации и их влияние на социально-экономическое развитие регионов. 2020.
3. Герасимова, Н. Социальные и экологические аспекты устойчивого развития. 2018.
4. Barbier, E. B. The Economics of Ecosystem Services and Biodiversity. Routledge. 2017

УДК 004.932

Е.В. Обухова, Н.П. Шутько
Белорусский государственный технологический университет
Минск, Беларусь

РАЗРАБОТКА МОДУЛЯ КОНВЕРТАЦИИ ВИДЕОРОЛИКОВ НА ОСНОВЕ БИБЛИОТЕКИ MOVIEPY и ФРЕЙМВОРКА FFMPEG

Аннотация. *В тексте рассматривается процесс проектирования и программной реализации модуля конвертации медиафайлов в формат OGV. Проведен сравнительный анализ двух подходов к реализации: с использованием Python-библиотеки MoviePy и посредством прямого взаимодействия с фреймворком FFmpeg. Описана архитектура разработанных решений. Приведены результаты эмпирического тестирования производительности.*

E.V. Abukhova, N.P. Shutko
Belarusian State Technological University
Minsk, Belarus

DEVELOPMENT OF A VIDEO CONVERSION MODULE BASED ON THE MOVIEPY LIBRARY AND FFMPEG FRAMEWORK

Abstract. *The text discusses the design and software implementation of a module for converting media files to the OGV format. A comparative analysis of two implementation approaches is performed: using the MoviePy Python library and through direct interaction with the FFmpeg framework. The architecture of the developed solutions is described. The results of empirical performance testing are presented.*

Ключевым этапом проектирования системы для работы с видеоконтентом, в частности плагина для транскодирования

видеофайлов в формат OGV, выступает разработка производительного модуля конвертации [1]. В настоящей работе рассматриваются два подхода к реализации данного модуля: использование библиотеки MoviePy и фреймворка FFmpeg, а также измерение скорости их работы. Фиксация данных метрик является подготовительным этапом, предваряющим дальнейшее исследование по выявлению наиболее эффективного инструмента для финальной реализации плагина. Краткая информация о каждой библиотеки представлена ниже.

Библиотека MoviePy [2] представляет собой инструмент, использующийся для обработки и редактирования видеофайлов. Она предоставляет пользователю широкий набор функций, позволяющих осуществлять чтение, модификацию и компоновку видео- и аудиоконтента, а также создавать собственные эффекты и анимации. Библиотека поддерживает работу с популярными видео-форматами, включая MP4, AVI, MOV и другими, что увеличивает ее популярность у пользователей. Также MoviePy отличается простотой интеграции за счёт интуитивно понятного API и возможностью объединения с другими библиотеками Python.

FFmpeg представляет собой универсальный фреймворк с открытым исходным кодом, предназначенный для манипуляций с аудиовизуальными потоками. Базируясь на библиотеках libavcodec и libavformat, данный пакет обеспечивает полную обработку медиаданных, в том числе декодирование, фильтрацию и финальное кодирование в различные форматы. Возможности FFmpeg охватывают задачи разной сложности, включая пакетное транскодирование, склейку видеофрагментов и потоковое вещание, что обуславливает его широкое применение в сфере разработки мультимедийных систем и профессионального видеопроизводства.

Рассмотрим реализацию модуля на основе библиотеки MoviePy. Код разработанного модуля можно разделить на четыре блока: получение данных, формирование видеопотока, обработка аудиопотока и финальная сборка.

Блок получения данных представлен методом convert_to_ogv, который представлен в листинге 1. Реализованный модуль спроектирован для работы с предварительно декодированными медиаданными, что позволяет отделить процесс чтения исходных файлов и их декодирования от процесса их кодирования. Входными параметрами функции выступают: последовательность видеокадров, полученных ранее, которые представлены в виде списка многомерных массивов Numpy, массив аудиосемплов, которые также были получены в результате работы других модулей программы, а также

частота дискретизации аудиосигнала. Такая архитектура обеспечивает универсальность модуля, позволяя ему обрабатывать данные, полученные из различных декодеров.

```
def convert_to_ogv(
        video_frames: list[np.ndarray],
        audio_data: np.ndarray,
        audio_samplerate: int,
        output_filepath: str,
        fps: int = 30):
    if not video_frames:
    print("Error: No video frames provided.")
    return False
```

Листинг 1 – Интерфейс функции и входные данные

Блок, отвечающий за формирование видеопотока представлен использованием класса ImageSequenceClip библиотеки MoviePy (листинг 2). Этот шаг является ключевым моментом перехода списка матриц пикселей, в которые был декодирован видеофрагмент, к единому видеообъекту. С помощью данного класса происходит привязка кадров путем задания целевой частоты кадров (параметр fps), что трансформирует статический массив данных в воспроизводимую последовательность изображений.

```
video_clip = ImageSequenceClip(video_frames, fps=fps)
```

Листинг 2 – Использование класса ImageSequenceClip

Блок, отвечающий за обработку и интеграцию аудиопотока представлен в листинге 3. Он включает в себя этап предварительной валидации и форматирования данных. Происходит проверка наличия аудиопотока и в случае успеха производится проверка размерности полученного массива Numpy. Также для совместимости с внутренними алгоритмами библиотеки MoviePy полученные одномерные массивы преобразовываются в двумерную структуру (N, 1), которая соответсвует монофоническому сигналу. После, на основе подготовленных данных инициализируется экземпляр класса AudioArrayClip, который впоследствии интегрируется в структуру видеообъекта с помощью метода .with_audio.

```
audio_clip = None
  if audio_data is not None:

    if audio_data.ndim == 1:
```

```
    audio_data_reshaped = audio_data.reshape(-1, 1)
  else:
    audio_data_reshaped = audio_data
  audio_clip = AudioArrayClip(audio_data_reshaped, fps=audio_samplerate)
  video_clip = video_clip.with_audio(audio_clip)
```

Листинг 3 – Обработка и интеграция аудиопотока

Блок, отвечающий за финальную сборку видео- и аудиопотоков целевой файл, представлен методом .write_videofile() (листинг 4). Для реализации записи в контейнер OGV для данного метода явно указываются кодеки libtheora для видеопотока и libvorbis для аудиопотока. Также в процессе экспорта для достижения лучшего качества целевого файла задаются параметры битрейта. Для достижения стабильности работы приложения при пакетной обработке файлов также был добавлен блок finally с методом .close(), который отвечает за освобождение ресурсов.

```
try:
    video_clip.write_videofile(
      output_filepath,
      codec="libtheora",
      audio_codec="libvorbis",
      fps=fps,
      bitrate="8000k",
      audio_bitrate="320k")
    return True

  except Exception as e:
    return False
  finally:
    if video_clip:
      video_clip.close()
    if audio_clip:
      audio_clip.close()
```

Листинг 4 – Финальная сборка потоков в целевой файл

В результате с использованием модуля конвертации, реализованном с помощью библиотеки Moviepy, полная обработка (декодирование, парсинг и кодирование) двух видеофайлов заняла 92 секунды.

Рассмотрим реализацию модуля с использованием фреймворка FFmpeg. Код разработанного модуля можно разделить на три блока: получение данных и их нормализация, сериализация аудиопотока и финальная сборка файла.

Первый блок отвечает за валидацию и препроцессинг входных видеоданных (листинг 5). Исходя из требований алгоритма цветовой субдискретизации OGV, реализовано автоматическое кадрирование изображения до ближайших четных значений ширины и высоты. Далее также все полученные данные приводятся к необходимым типам: значения пикселей, представленные в формате с плавающей точкой, нормализуются и конвертируются в целочисленный формат uint8 (диапазон [0, 255]), что является стандартом для передачи данных в RGB-потоке.

```
h, w = video_array.shape[1], video_array.shape[2]
  new_h = h - (h % 2)
  new_w = w - (w % 2)
  if new_h != h or new_w != w:
    video_array = video_array[:, :new_h, :new_w, :]
    h, w = new_h, new_w

  if video_array.dtype != np.uint8:
    if np.issubdtype(video_array.dtype, np.floating):
      video_array = (video_array * 255).clip(0, 255).astype(np.uint8)
    else:
      video_array = video_array.astype(np.uint8)
```

Листинг 5 – Валидация и препроцессинг входных видеоданных

Второй блок, отвечающий за работу со звуком, представлен в листинге 6. Для корректной обработки звукового сопровождения используется механизм промежуточной сериализации. Так, входной массив аудиосэмплов преобразуется из формата с плавающей точкой в 16-битный формат PCM (Pulse Code Modulation), после чего сохраняется во временный файл-контейнер WAV с использованием стандартной библиотеки wave. Это позволяет избежать ошибок синхронизации и переполнения буфера при последующем мультиплексировании.

```
has_audio = False
  if audio_array is not None and audio_array.size > 0:
```

```
    has_audio = True
    if audio_array.ndim == 1: audio_array = audio_array[:, np.newaxis]
                audio_int16 = (audio_array * 32767).clip(-32768,
32767).astype(np.int16)
    with wave.open(temp_audio, 'wb') as wf:
      wf.setnchannels(audio_array.shape[1])
      wf.setsampwidth(2)
      wf.setframerate(audio_samplerate)
      wf.writeframes(audio_int16.tobytes())
```

Листинг 6 – Обработка аудиопотока

Процесс создания конечного файла можно разделить на два этапа: генерация промежуточного файла и финальное транскодирование в OGV. Результатом первого этапа является созданный временный видеофайл, который служит надежным источником данных для финального кодирования, минимизируя нагрузку на оперативную память при обработке больших массивов. Заключительный этап конвертации транскодирует промежуточные медиафайлы в целевой формат OGV. Для управления параметрами сжатия кодека libtheora (видео) и libvorbis (аудио) формируется командная строка вызова FFmpeg. Код транскодирования представлен в листинге 7.

```
cmd_phase1 = [
    ffmpeg_cmd, '-y', '-loglevel', 'error',
    '-f', 'rawvideo', '-vcodec', 'rawvideo',
    '-s', f'{w}x{h}', '-pix_fmt', 'rgb24', '-r', str(video_fps),
    '-i', '-',
    '-c:v', 'huffyuv',
    temp_intermediate]
 cmd_phase2.extend([
    '-c:v', video_codec,
    '-b:v', '5000k',
    '-maxrate', '7000k',
    '-bufsize', '14000k',
    '-pix_fmt', 'yuv420p',
    '-flags', '+qscale',
  ])
```

Листинг 7 – Транскодирование медиафайлов в формат OGV

В результате использования, разработанного с помощью библиотеки Moviepy модуля конвертации, полная обработка (декодирование, парсинг и транскодирование) двух видеофайлов заняла 53 секунды. Однако из-за увеличенной сложности реализации полученные итоговые файлы уступают в качестве файлам, созданным с помощью библиотеки MoviePY.

Список использованных источников

1. Обухова, Е. В., Н. П. Шутько Сравнительный анализ алгоритмов видеокодирования THEORA, MPEG-4 и H.263 / Е. В. Обухова, Н. П. Шутько // Сборник материалов международной молодежной научно-практической конференции студентов, аспирантов и молодых ученых, Махачкала, 19–20 ноября 2024 г. – Махачкала, Типография ФОРМАТ, 2024. – С. 71-73.

2. Журавлев, А. А. Сравнительный анализ эффективности программных инструментов для разбивки видео на кадры на примере области оценки качества дорожной поверхности / А. А. Журавлев, К. А. Аксенов // ИВД. – 2024. – №3 (111).

УДК 004.822

Е.В. Обухова
Белорусский государственный технологический университет
Минск, Беларусь

ВЛИЯНИЕ СТРУКТУРНЫХ ПАРАМЕТРОВ НА ЭФФЕКТИВНОСТЬ СИНХРОНИЗАЦИИ ДРЕВОВИДНЫХ МАШИН ЧЕТНОСТИ В УСЛОВИЯХ СЕТЕВЫХ ЗАДЕРЖЕК

Аннотация. *В работе исследуется влияние структурных параметров древовидных машин четности (ДМЧ) на эффективность процесса синхронизации. На основе полученных экспериментальных данных построены зависимости количества итераций и скорости генерации ключа от конфигурации нейросети. Сформулированы рекомендации по выбору оптимальной конфигурации ДМЧ .*

E.V. Abukhova
Belarusian State Technological University
Minsk, Belarus