Санкт-Петербургский государственный университет
Направление: 01.03.02 Прикладная математика и информатика

ООП: Прикладная математика, фундаментальная информатика и программирование

ОТЧЕТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

Тема работы: *автоматическое обнаружение мерцающих сцен в видеоряде*

Выполнил: Семенов К. Е. студент гр. 21.Б09
Научный руководитель: Коровкин М. В. доцент, к. ф.-м. н.

Санкт-Петербург
2024

Содержание

Введение

В предыдущей части работы над темой был реализован алгоритм, позволяющий обнаружить мерцающие кадры в видеоряде. Результаты, полученные в ходе его тестирования, оказались неудовлетворительными с точки зрения производительности.

Задачей в нынешней части работы является оптимизация его работы для возможности определения мерцающих последовательностей в реальном времени или с небольшой задержкой, то есть время работы алгоритма не должно превышать продолжительности видеоролика.

Для этого может понадобиться обрабатывать входной видеоряд другим способом, а также задействовать больше ресурсов устройства, на котором программа будет исполняться.

План работы:

1. Определение оптимального способа обработки входного видеопотока
2. Ознакомление с возможностями библиотеки OpenCV по ускорению обработки видео
3. Реализация оптимизированного алгоритма
4. Тестирование оптимизированной версии программы и сравнение со старой версией по результатам и производительности

Подготовительная часть

Цифровое видео представлено своим файлом-медиаконтейнером, в котором хранятся закодированные аудиопотоки, видеопотоки и метаданные, такие как разрешение видео и информация о кодировке. Видеопоток обработан алгоритмом сжатия, таким как, например, H.264. До кодирования видеопоток представляет из себя трехмерную информацию по каждому пикселю в определенной цветовой схеме.

Наибольший интерес сейчас представляет случай, когда исходный видеопоток имеет цветовую модель YUV (YcbCr в цифровом случае), поскольку в этом случае компонента яркости Y (luma) уже посчитана для каждого кадра видео, и таким образом можно убрать функцию ее вычисления из программы. К счастью, YCbCr является стандартом для представления кадров на многих интернет-ресурсах, например, на крупнейшем видеохостинге Youtube и стриминговой платформе Twitch.

Для экономии памяти при кодировании видеопотока используется метод цветовой субдискретизации (Chroma subsampling). При котором компоненты цветности хранятся не попиксельно, а как среднее соседних. Однако яркостная компонента цвета хранится для каждого пикселя всегда, поэтому такое кодирование никак не мешает ее получению.

Для получения субдискретизированного видеопотока можно использовать популярный инструмент **FFmpeg**. Утилита предоставляет возможность декодирования исходного видеопотока из медиаконтейнера, создавая байтовую последовательность в выбранном пользователем формате субдискретизации (флаг **`-pix_fmt`**). Причем FFmpeg позволяет задействовать возможности процессора, на котором она запущена, такие как аппаратные средства декодирования и использование параллелизма на уровне инструкций и задач, что является большим плюсом с точки зрения производительности.

Обработка субдискретизированного видеопотока

Самый часто встречающийся способ субдискретизации — это схема 4:2:0 planar. По этой схеме будем получать байтовую последовательность. Эта схема представляет собой способ кодирования 12-ю битами на пиксель. Расположение байтов на рисунке 1.

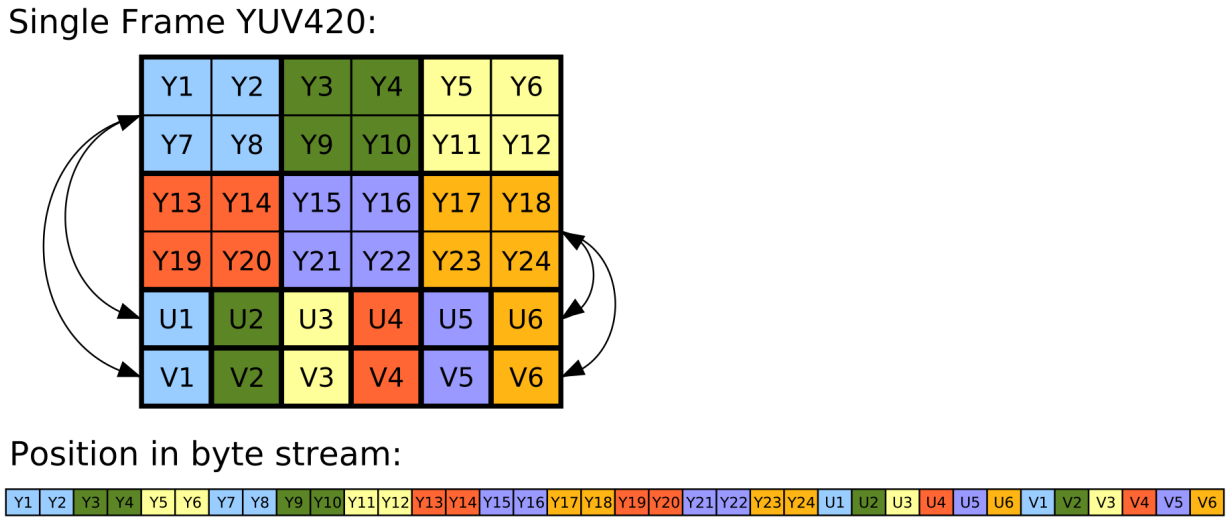

Рисунок 1: расположение байтов компонент цвета в памяти при субдискретизации yuv420p
[источник рисунка (Wikipedia)]

Теперь с известным расположением остается только заносить данные в буфер, размер которого равен числу пикселей в кадре.

Уменьшение размера видео

Так как для алгоритма обраружения вспышек важно определить долю экрана, охватываемую вспышкой, а не точное расположение каждого пикселя, который менял яркость, то можно исходный видеофайл обрабатывать в меньшем разрешении для значительного ускорения работы программы. FFmpeg также предоставляет функции снижения разрешения видео.

Ускорение работы в OpenCV

Библиотека OpenCV предоставляет фреймворк **`parallel_for_()`** для использования фреймворков для параллельных вычислений, таких как Intel Threading Building Blocks, OpenMP и системных потоков, как POSIX-потоки, Apple GCD, Windows RT Concurrency. **`parallel_for_()`** использует их в порядке доступности, так что это решение должно ускорять

выполнение на всех основных платформах. Большим плюсом является то, что распараллеливание процессов производится автоматически и остается только написать тело цикла, мало отличающееся от скалярного варианта.

Описание оптимизированного алгоритма

Предобработка.

Создаем экземпляр **cv::VideoCapture**, проверяем инициализацию экземпляра методом **isOpened()**. Получаем ширину **initHeight**, высоту **initWidth** исходного видео, соотношение сторон ratio, число кадров в видео **totalFrames**.

Проверяем размеры: если **initHeight** > 640, то будет происходить изменение разрешения видео. Для этого используем новые значения **height**, **width**, вычисляемые по формулам:

1. Если **initHeight**>**initWidth**, то **width** = 640, **height** = **width**/**ratio**
2. Если **initWidth**>**initHeight**, то **height** = 640, **width** = **height***ratio**

Создаем буфер типа **char** с размером **frameSize**, равным произведению **height***width**. Вызываем ffmpeg:

```
ffmpeg -i [файл] -filter:v scale=[height]:-1 -c:a copy out.mp4
```

Далее вызываем ffmpeg для получения декодированной последовательности значений YCbCr:

```
ffmpeg -i out.mp4 -c:v rawvideo -pix_fmt yuv420p out.yuv
```

Если изменения размеров не произошло, то есть **initHeight**<640, то эту же команду вызываем для исходного файла, а не уменьшенного.

Обработка байтовой последовательности.

Производится в цикле на **totalFrames** итераций.

1. Чтение в буфер **frameSize** байтов.
2. Создание матрицы яркостей $\mathbf{L_k}$ как класса **cv::Mat** типа **CV_8UC1** (одноканальный 8-бит массив) с указателем на данные буфера.
3. Создание в теле **parallel_for_()** матрицы разницы яркостей $\mathbf{L_{diff}} = \mathbf{L_k} - \mathbf{L_{k-1}}$ и разделение ее на положительные и отрицательные матрицы приращений $\mathbf{L_+}$ и $\mathbf{L_-}$

4. Преобразуем матрицы $\mathbf{L_+}$ и $\mathbf{L_-}$ в массивы $\mathbf{h_+}$ и $\mathbf{h_-}$ размером [$[1 \times frameSize]$Сортируем в порядке убывания, и определяем, есть ли ненулевой элемент на индексе $\frac{1}{4} frameSize$.
    - Если так для *обоих* массивов, то считаем среднее у каждого и среднее изменение яркости **`lumDiff`** будет наибольшим из средних значений по каждому массиву.
    - Если есть нулевой элемент на индексе $\frac{1}{4} frameSize$ *у одного* из массивов, то возвращаем среднее того, у которого элемент на этом индексе ненулевой.
    - Иначе возвращаем 0.
5. Считаем среднее **`lumMeanCurr`** яркости кадра.
    - Если минимальная средняя яркость на (k-1)-м и k-м шагах (**`lumMeanCurr`** на k-м, **`lumMeanPrev`** на (k-1)-м шаге) <=160 нит и **`lumDiff`** >=20 нит, то получена вспышка, в конец вектора индексов вспышек добавляем номер кадра.
    - Иначе получено приемлемое изменение яркости и переходим к (k+1)-му шагу.

Тестирование и оценка производительности.

Сравнение на тех же видеофайлах старой и новой версий программы.

№	$T_{видео}$, с	*height x width*	FPS	$T_{стар}$, с	$T_{нов}$, с	$N_{всп\_стар}$	$N_{всп\_нов}$
1	30	1920x1080	30	~123	~6.5	0	0
2	79	480x360	30	~31	~18	1422	1504
3	31	480x360	25	~8	~3.9	12	12
4	12	1280x720	30	~20	~2.6	93	93

Видео 1 (только скачивание), Видео 2 (youtube), Видео 3 (youtube), Видео 3 (youtube)

Вывод из проведенной работы

Работа по оптимизации производительности алгоритма проделана успешно. На видео с большим разрешением это особенно заметно, что и требовалось получить на данном этапе. Небольшие расхождения по подсчету вспышек в одном из тестов можно объяснить тем, что компонента яркости считается в видео с немного другими весовыми коэффициентами. Дальнейшие шаги по развитию темы могут включать в

себя улучшение пользовательского взаимодействия, например, создание графического интерфейса. Также планируется добавить режим работы в реальном времени, так как производительность работы алгоритма позволяет это сделать теперь, а также вывод исправленного видео без мерцающих последовательностей.

Ссылки и литература

- Документация OpenCV: https://docs.opencv.org/4.x/
- «Guidance for the reduction of photosensitive epileptic seizures caused by television», ITU-R BT.1702-3, 2023
- Статья «Automatic detection of flashing video content», Lúcia Carreira, Nelson Rodrigues, Bruno Roque, Maria Paula Queluz, 2015
- Отчет о научной работе по этой теме за прошлый семестр, автор Семенов К. Е., научный руководитель Коровкин М. В., 2023
- Статья в Wikipedia про цветовое пространство YCbCr
- Спецификация FOURCC по пиксельным форматам YUV
- Документация ffmpeg: https://ffmpeg.org/ffmpeg.html