Efficient Video Editing for Mobile Applications

Ignasi Vegas Pajaro
Department of Computer Science
Manhattan College
New York, USA

Ankur Agrawal
Department of Computer Science
Manhattan College
New York, USA

Tina Tian
Department of Computer Science
Manhattan College
New York, USA

***Abstract*—Recording, storing and sharing video content has become one of the most popular usages of smartphones. This has resulted in demand for video editing apps that the users can use to edit their videos before sharing on various social networks. This study describes a technique to create a video editing application that uses the processing power of both GPU and CPU to process various editing tasks. The results and subsequent discussion shows that using the processing power of both the GPU and CPU in the video editing process makes the application much more time-efficient and responsive as compared to just the CPU-based processing.**

***Keywords*—iOS programming; Image processing; GPU; CPU; Objective-C; GPUImage; OpenGL**

I. Introduction

Smartphones have become an essential part of our day-to-day life. Americans spend about one hour a day on their smartphones using mobile applications [1]. The iPhone is the most used device, occupying 47% of the smartphone market share [2].

We consume different types of content on our smartphones such as news, social-media, images, video games, music, films, TV shows, etc. Especially, the number of video content distributed around the Internet is growing exponentially every year due to popular video hosting platforms like YouTube, Facebook, Snapchat and Instagram. The consumption of video in mobile platforms is expected to grow 67% year-on-year until 2019 [3] as can be seen in Fig 1.

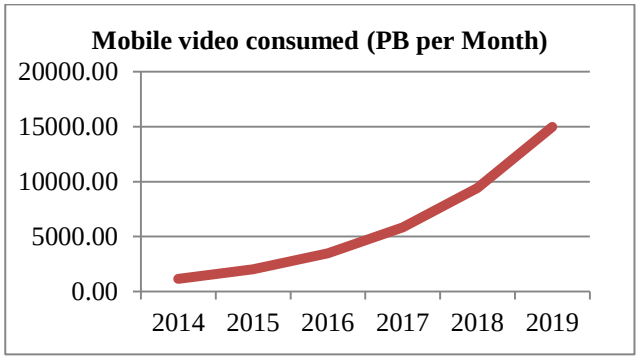

Fig. 1. Evolution of Mobile video consumed in PB per month

As a result of the high quality camera in iPhones, we can record video in high quality with a device that is always in our pocket. The videos can then be shared with our friends across different social-media platforms. With more and more videos being recorded and shared, it has become important for the users to be able to edit those videos before being published on the Internet. Video editing is the process of manipulating video images, adding audio and/ or visual effects. Since smartphones are getting more and more powerful with each passing day in terms of processing and memory, it is possible to build iPhone applications to edit videos that the users record, without the need of a computer and with a better and faster user experience.

This paper presents a study on developing a video editing application for iOS platform. The application uses image processing algorithms and iOS programming techniques. Image processing is the processing of images using mathematical operations by using any form of signal processing for which the input is an image, a series of images, or a video and the output may be either an image or a set of characteristics or parameters related to the image. iOS programming techniques use a set of libraries, algorithms and best practices that are used to build iPhone applications.

This application allows the user to record a video or to import a video stored in your iPhone camera roll. The user can select a specific part of the video and crop the video if it is required. The user can then add some image filter effects along with a background song. Finally, the user can save the resulted video back to the iPhone.

II. Methods

A. Technologies used

The application is programmed in iOS version 9.0[4]. iOS version 9.0 runs in 80% of the iOS devices using xCode version 7.3 [5] and Objective-C [6] as language development. Recently, Apple launched a new programming language for iOS called Swift [7]. This application however is programmed in Objective-C instead of Swift since Objective-C is a more evolved language with more documentation about video processing than Swift.

B. Libraries used

For the entire iOS application flow and user interface, we have used the Cocoa Framework [8], a group of native libraries provided by Apple to create the user interface of an application.

The video capture, video importing/exporting and video cropping, is implemented using UIImagePickerController [9]. This is a class created by Apple to work with media files.

The video filter processing is created using GPUImage [10], a third-party library created by Brad Larson. This library gives you the opportunity to use the GPU to process the video instead of CPU. The video processing tools provided by Apple

only allows to process video using CPU. Also, using GPUImage you can use predefined filters or you can create filters of your own.

To preview the video, the application uses Core Image [11], an iOS native library that allows you to reproduce media content in your application.

AVFoundation [12] is used to add custom background audio to the videos. This is a native iOS library provided by Apple to manipulate audio in media files.

C. Views

In iOS, when we talk about a view, we are referring to a screen in the application. Our application has four different views, as discussed below.

The first view allows the user to select a video for editing. The user can select between recording a video using the iphone camera and importing a video from the iPhone camera roll. The user can also select certain parts of the video to be processed, and delete the rest of the video. The new video segment, thus created, is saved in a temporary directory inside the application.

Once the video is selected for editing, the filter screen appears. This view provides a preview of the video where the user can select a filter to apply. There is an option to keep the video as it is without applying any filters. When a filter is selected, the application sends the video to the GPU. This means that the CPU is not processing the video, as the GPU works as a separate thread. While the video is being processed, a loading icon is displayed. When the process is complete, the processed video can be viewed with the filter applied. If the user does not like the applied filter, they can select another filter and the above process will be repeated. When the video has been processed, it remains in the temporary directory.

The third view is the audio view. This view shows a classical iOS TableView [13] with a list of all the available songs that can be chosen for the video. The song files are stored with the application, as the application only offers a few songs and the durations are not longer than twenty seconds. When the user selects a song, the video is processed again. The processing uses the CPU by creating a parallel thread, so now the application continues to run in the main thread. The user also has the option to not add any song to the video. The video is again saved in the temporary directory after an audio song has been added to the video.

The fourth view offers a final preview of the video with the new audio included. Here, the user has the option to save the video to the camera roll. Note that, so far, the video is only stored in a temporary folder. This is being done to prevent unnecessary use of memory space and CPU as it is more efficient to work with a file stored in a temporary directory inside the application space.

D. Filters

GPUImage works on top of OpenGL shaders[14]. OpenGL Shaders are programs designed to run on some stage of a graphic processor (GPU). As a result, our application can process videos using GPU and also use predefined image filters or create a custom filter using OpenGL features.

As mentioned earlier, when the application starts processing the video, the CPU creates a parallel thread. This parallel thread is then processed by the GPU as shown in Fig. 2. The GPU reads every frame of the video and processes each frame separately. When all the frames are processed, the GPU returns the control back to the CPU.

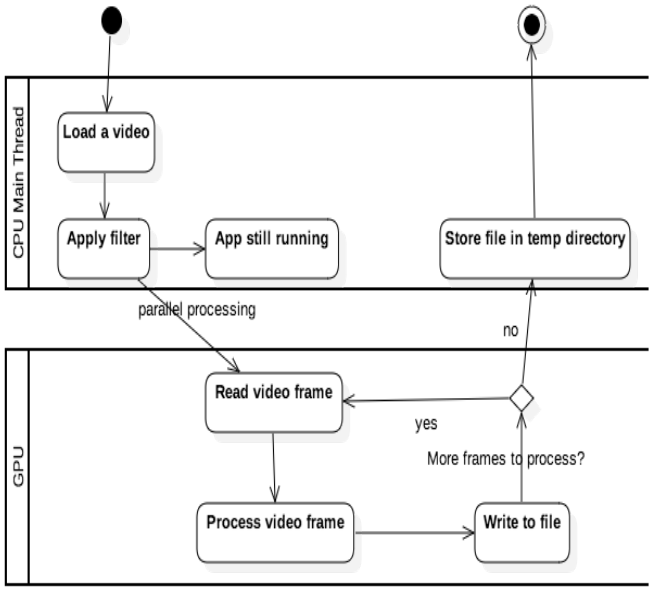

Fig. 2. GPU and CPU state while processing a video

The process that OpenGL Shaders use to process an image is called rendering pipeline. The OpenGL rendering pipeline defines a number of stages for this shaders as shown in Fig. 3.

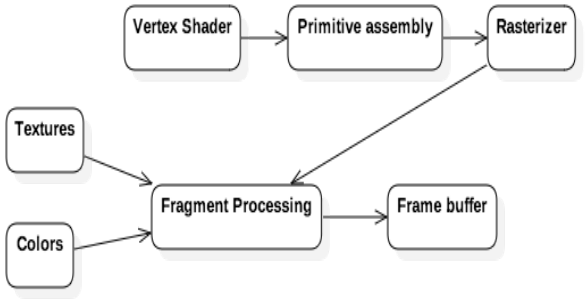

Fig. 3. States of the rendering pipeline

The vertex shader [15] transforms the video images into individual vertices. Primitive assembly [16] connects the vertices created by the vertex shader to form figures which are called primitive units. Rasterization [17] is then applied, which transforms the primitive units into smaller units called fragments. In the fragment processing stage [18], colors and textures are applied to the fragments, which is then saved in a Frame Buffer [19]. The frame buffer allows us to create an image or show the image on a screen. The key advantage of using OpenGL Shaders is that the various operations can be run in parallel in the GPU allowing for a more responsive application.

III. RESULTS

Fig. 4 displays the first view of the application. In this view, you can select two options; Record a video using the iPhone camera or import a video from the camera roll. Fig. 5 shows the view where the user can crop the video. Fig. 6 shows the view where a filter can be applied to the video. The application currently provides 15 popular filters, as shown in Table I.

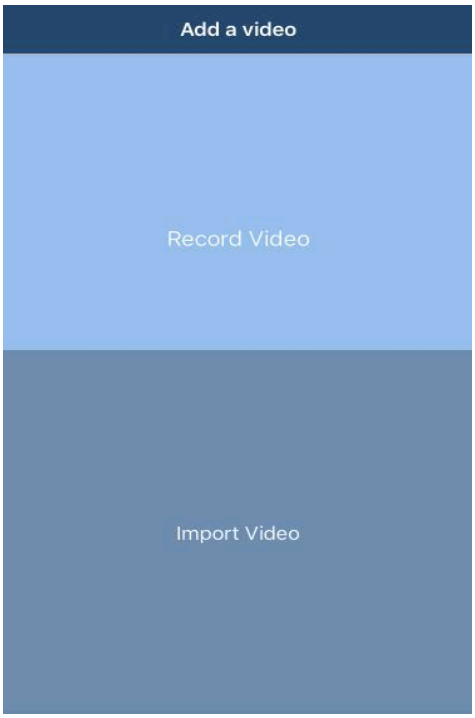

Fig. 4. First app view with two available options

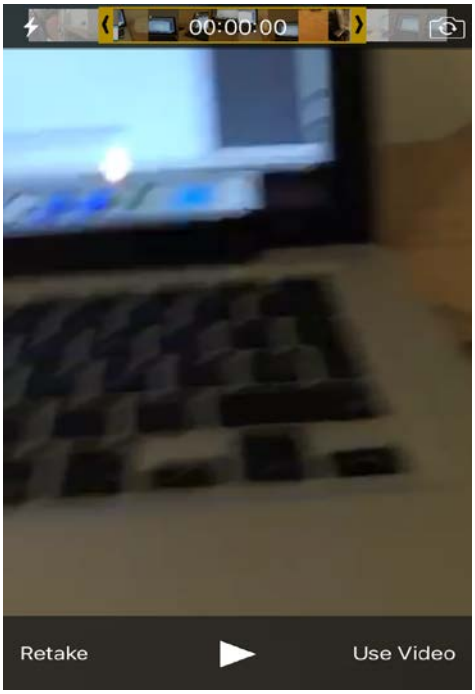

Fig. 5. Second app view to crop the video

Fig. 6. Third app view to apply filters

TABLE I. FILTERS AVAILABLE

Sepia	Blur	Color Space
Color Invert	Sobel Edge	Emboss
Errosion	Exposure	Gamma
Laplacian	Luminance	Posterize
Prewitt Edge	Saturation	Gaussian

Fig. 7 provides a view where the user can choose an audio song that will be added to the video. Currently, the application provides 10 audio songs. These songs have been downloaded from jammendo.com [20], which are under Creative Commons [21] license. The last view, as shown in Fig. 8, provides a preview of the processed video and gives user the option to save the video on the camera roll.

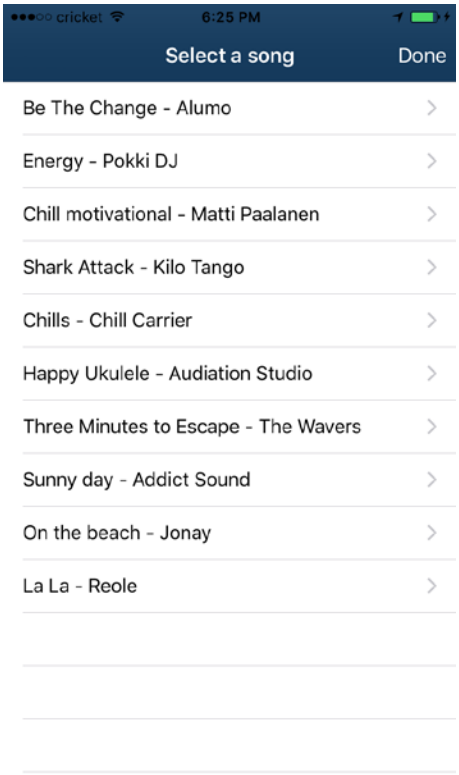

Fig. 7. Fourth app view to select an audio song

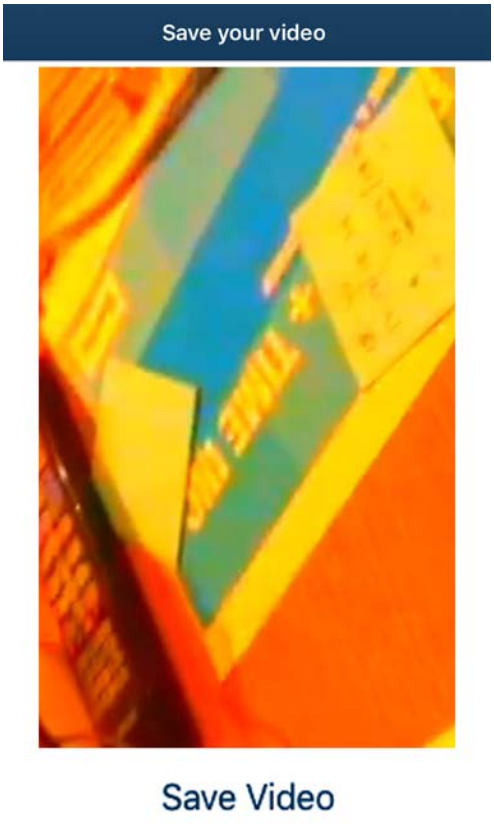

Fig. 8. Fifth app view to save the edited video

IV. Discussion

A. Using GPU or CPU for image processing

In the methods section, we mentioned about using GPU processing alongside the CPU processing for many of the processing tasks. For parallel operations like video processing, using GPU has significant performance advantages over CPU. The GPUImage framework takes only 2.5 ms on an iPhone 4 to upload a frame from the camera, apply a gamma filter, and display, versus 106 ms for the same operation using Core Image and 460 ms using CPU-based processing. This makes GPUImage 40X faster than Core Image and 184X faster than CPU-based processing. On an iPhone 4S, GPUImage is 4X faster than Core Image for this case, and 102X faster than CPU-based processing. [22].

CoreImage is the library provided by Apple to process images and video files. In newer devices like the iPhone 6, we can achieve the same performance using CPU or GPU. However, for this study we decided to use GPU processing because older devices like the iPhone 4 and iPhone 5 are more responsive when we utilize both GPU and CPU for video editing tasks.

B. Duration of the videos

Several social networks such as Instagram [23] and Snapchat [24] limit the length of videos that can be uploaded to 10 or 15 seconds. When a user uses a mobile application, they want a fast, responsive and a seamless user experience, and processing a video longer than 20 seconds can take a longer time thus negatively impacting the user experience. So, we decided to limit the duration of the videos that the users can take using the application to 20 seconds. Table II shows the video processing time using the application with different video durations. All videos are in 1080p with 30 frames per second. For this experiment, the blur effect was applied using an iPhone 6.

TABLE II. 1080p Video Processing Time on an Iphone 6

Length of video (seconds)	Video processing time (seconds)
10	3
20	7
30	10
40	14
50	17
60	21
70	25

Table III shows the video processing time using the application for videos of different durations in 640p (unlike videos in Table II that are in 1080p).

TABLE III. 640P VIDEO PROCESSING TIME ON AN IPHONE 6

Length of video (seconds)	Video processing time (seconds)
10	2
20	4
30	6
40	8
50	11
60	13
70	14

Table IV shows the video processing to apply blur effect using the application on iPhone 5s. All videos are in 1080p with 30 frames per second. As the data shows, the processing time required by an iPhone 5s is almost the double of the time required by iPhone 6 as shown in Table II. This is as a result of the iPhone 5s GPU being half as powerful as the iPhone 6 GPU [25].

TABLE IV. 1080P VIDEO PROCESSING TIME ON AN IPHONE 5S

Length of video (seconds)	Video processing time (seconds)
10	10
20	20
30	30
40	39
50	49
60	60
70	68

C. Future Work

Future work will involve improving the scalability of the application. For instance, we will have the songs list stored on the server. The application will be able to connect to the server and the songs can be downloaded to the smartphone. Another new feature will involve adding the option to select between different image qualities for the output video. With lower quality export videos, the processing will be faster as compared to a video generated using the high quality option.

V. CONCLUSION

Mobile applications and video content have become an integral part of our lives. It has become common for people to use their mobile phones to record, edit and share videos on social networking sites. This paper presents a video editing application for iOS devices that can be used to record videos and edit them. The edit features include cropping, applying filters or adding background audio. The application describes a technique to use the processing power of both the GPU and the CPU to improve the response time. The results show that using the processing power of the GPU alongside CPU in the video editing process makes the application more efficient and responsive.

REFERENCES

[1] Smart Insights http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/

[2] Nielsen Smartphones market share, http://www.nielsen.com/us/en/insights/news/2015/tops-of-2015-digital.html

[3] Cisco Visual Networking Index, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

[4] Apple iOS 9 Specifications, https://developer.apple.com/library/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS9.html

[5] Xcode at a glance, https://developer.apple.com/library/ios/documentation/ToolsLanguages/Conceptual/Xcode_Overview/

[6] About Objective–C, https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

[7] The Swift Programming Language, https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/

[8] Cocoa Touch Documentation, https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Cocoa.html

[9] UIImagePickerController Class Reference, https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIImagePickerController_Class/

[10] GPUImage, https://github.com/BradLarson/GPUImage

[11] Core Image Programming Guide, https://developer.apple.com/library/mac/documentation/GraphicsImaging/Conceptual/CoreImaging/ci_intro/ci_intro.html

[12] AVFoundation Programming Guide, https://developer.apple.com/library/ios/documentation/AudioVideo/Conceptual/AVFoundationPG/Articles/00_Introduction.html

[13] UITableView Class Reference https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableView_Class/

[14] Shaders, https://www.opengl.org/wiki/Shader

[15] Vertex Shader, https://www.opengl.org/wiki/Vertex_Shader

[16] Primitive Assembly, https://www.opengl.org/wiki/Primitive_Assembly

[17] Rasterization, https://www.opengl.org/wiki/Rasterization

[18] Fragment Processing, https://www.opengl.org/wiki/Framebuffer_Object

[19] Frame Buffer, https://www.opengl.org/wiki/Framebuffer_Object

[20] Jammendo Music, https://www.jamendo.com/?language=en

[21] Creative Commons, https://en.wikipedia.org/wiki/Creative_Commons

[22] GPUImage Overview, https://github.com/BradLarson/GPUImage

[23] Instagram FAQ, https://help.instagram.com/270963803047681

[24] Snapchat, https://support.snapchat.com/en-US/ca/snaps

[25] Apple A8 Chip, https://en.wikipedia.org/wiki/Apple_A8