Google Summer of Code - 2019

Final Report

Swaraj Hota (@Swaraj1998)
<swarajhota353@gmail.com>

Project: HEIF Support in FFmpeg
Mentor: Carl Eugen Hoyos (@cehoyos)

Description: The aim of this project was to add support for HEIF/HEIC files in FFmpeg. High-Efficiency Image File Format (HEIF) specifies the storage of individual images, image sequences, and their metadata into a container file conforming to the ISO Base Media File Format (ISOBMFF). This format has increasing usage in mobile devices.

Commits during Qualification and Work Period:
https://github.com/FFmpeg/FFmpeg/commits?author=Swaraj1998

Folder containing the patches produced during Work Period:
https://drive.google.com/drive/folders/1_IY1AajWzsIvAgPB8RmvM2DSrJzlzJz-?usp=sharing

Updated and improved commit after Work Period:
https://github.com/Swaraj1998/FFmpeg/commit/9a885cddb3550ab863a60d02c5fb78e4ae206cf1

Work Summary:
I was completely new to the domain of multimedia programming in the beginning. I started by solving the qualification tasks. These tasks led me to dig deeper and learn more about the field. Eventually, I also started getting acquainted with the large codebase. I was able to add support for two container formats, one of which required reverse engineering sample files which I really enjoyed doing. Both of these patches are merged.

Before beginning the project, I had to study the HEIF specifications in detail. Then I wrote a basic HEIF muxer which helped in testing and also in writing the required HEIF demuxer. I was able to write a demuxer that supported reading non-tiled images by the second phase.

The challenge was then to support tiled images as well, the final implementation of which was not very clear. Different developers in the community had different views on how it should be

implemented, and all were valid in their own way. It was really difficult to choose one over the other, so my mentor and I finally had to settle on some middle ground that worked for everyone. The idea was to implement two of the solutions, and provide options for the user to choose which way they get the final image.

One of the solutions was to decode the tiles inside libavformat, place these tiles in a larger frame (the final image), and send this decoded “raw video” frame to the calling application. This was a (relatively) simpler solution, and was a solution that can be used in various other “tiled” formats which were still unsupported due to the lack of a standardized way to handle tiling in FFmpeg. I successfully implemented this solution, and also added an option to disable decoding inside libavformat. If the option is set, the demuxer currently only sends the (encoded) tiles as different frames in a single stream. The decoding and stitching of these tiles could then be handled by the calling application.

For the calling application to be able to correctly handle tiles (in case decoding inside libavformat was disabled), we need to export some essential information about the tiles (# of tiles, # of rows and columns in the image grid, current tile #, etc.) along with the frames as “side data”. This was another proposed solution to handle tiling.

Current Status and Future Work:
The initial goal of the project, as mentioned in FFmpeg’s GSoC ‘19 page (here), is successfully completed. The demuxer is able to read and decode all the sample files I have. Unfortunately, I was not able to send the patch to the mailing list before completing GSoC, but I will soon send the patch after refining and adding more features, as my mentor also suggests.

The demuxer can be improved further to support the full range of features that HEIF format has to offer, but as most of the features are not currently in use anywhere, this is a low priority. The support for “side data” solution as mentioned before, however, must be implemented. I am planning to add this myself after the completion of GSoC. Further, the less important HEIF “muxer” can also be improved a bit before it can be merged. I am planning to do this too.

Final Words:
The past three months have been excruciating and also thrilling at the same time. There were times when I thought “that’s it”, and was stuck for days on end having no idea what to do. HEIF, being an unusually complicated container format for an “image”, was considered difficult to be implemented in FFmpeg (also because of a very different internal structure of FFmpeg). But after a lot of head banging, days of debugging, lots of code staring, and most importantly, a lot of help from my mentor and other developers in the community, I was finally able to complete this in time. I would like to thank all of them, especially my mentor, who had to endure my silliest questions and always pointed me in the right direction. I had a wonderful experience at FFmpeg and would like to continue contributing in the future.