Assignment 2 - Audio and Video Coding

Pedro Alagoa [72534] Gonçalo Grilo Alexandre [72608]

I. Summary

This report describes the implementation and design of a video player, encoder and decoder developed for the class. The purpose of the tool is to play YUV files, as well as encoding a new version of the file using less bits, while maintaining the same data (lossless encoding).

II. Running instructions

III. External libraries

This tool uses the **Program options** library of the Boost C++ Libraries [1].

IV. Practical work

A. Video Player

The developed player supports 3 types of chroma subsampling, 4:4:4, 4:2:2 and 4:2:0. The data in the video file is stored in the planar format, which can be seen in Figure 1.

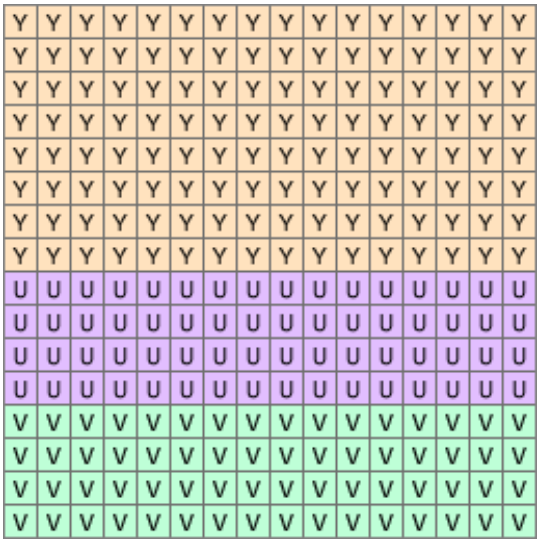

Fig. 1: Planar storing format.

To get the correct YUV values for different chroma subsamplings, we need to read the file using specific calculations for the indexes:

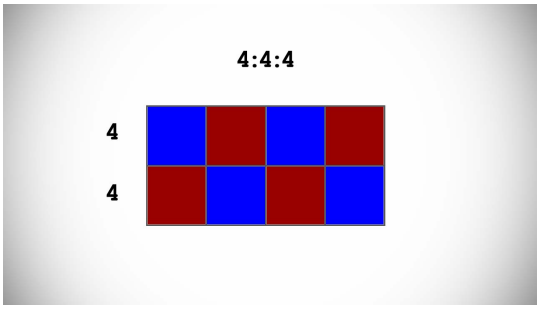

Fig. 2: 444.

```
y = imgData[i/3];
u = imgData[(i/3) + (yRows * yCols)];
v = imgData[(i/3) + (yRows * yCols) * 2];
```

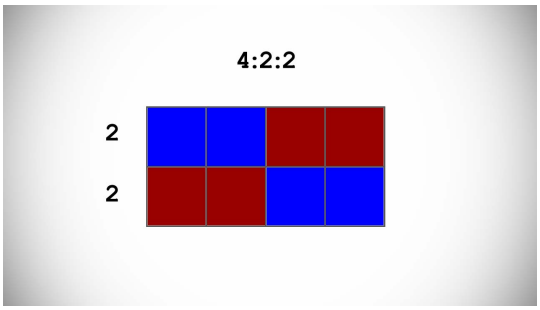

Fig. 3: 422.

```
y = imgData[i];
u = imgData[(yRows * yCols) + (i/2)];
v = imgData[(yRows * yCols)*3/2 + (i/2)];
```

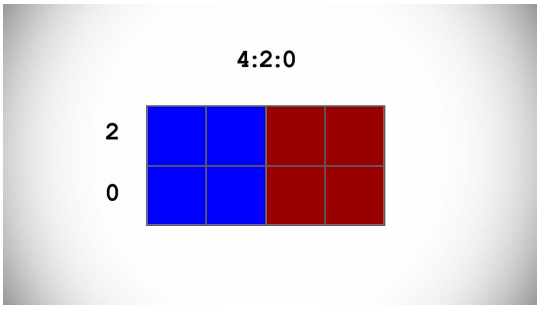

Fig. 4: 420.

```
y = imgData[i];
int nRow = i/yCols/2;
u = imgData[i/2%yCols + ((nRow-(nRow%2))/2)*
    yCols + (yRows * yCols)];
v = imgData[i/2%yCols + ((nRow-(nRow%2))/2)*
    yCols + (yRows * yCols)*5/4];
```

After retrieving the YUV values, they are converted to RGB using the traditional formulas. Afterwards,

clipping is applied, so that they remain values that range from 0 to 255.
Finally, they are copied to the OpenCV buffer where will eventually be shown along with the other pixels of the same frame.

B. *Intra frame lossless encoder*

For this encoder we used a simplified version of the **JPEG-LS**. In our version of this algorithm the predictor consists of a fixed and a adaptative component like in [2].

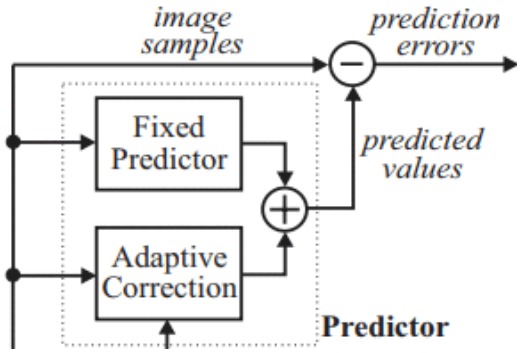

Fig. 5: Predictors used

The fixed predictor will make a simple test to detect horizontal and vertical edges and make a guess using only 3 near pixels:

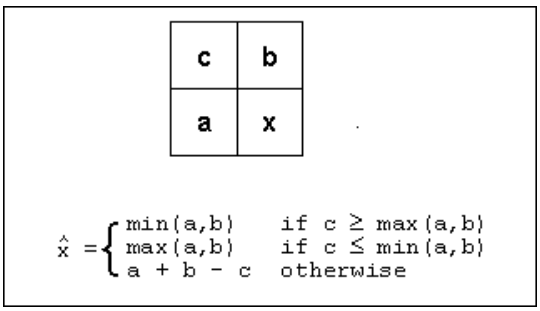

Fig. 6: Fixed predictor used.

The adaptative part of the predictor is context dependent and will produce a additive term. This term will try to aproach the previous prediction error value to 0, by using the accumulated value of errors in the current context. This value is based on a context that is tuned during the encoding. The context is conditioned by the local gradient calculated with:

$$g_1 = D - B$$
$$g_2 = B - C$$
$$g_3 = C - A$$

Fig. 7: Gradient calculation.

The residuals produced by this predictors are then encoded using Golomb with a value of $\mathbf{m} = 2^k$ as in [2]. The parameter **k** is calulated for each context using: $k = min(k'|2^{k'}N \geq A)$ where N is the number of occurrencies of the current context and A is the accumulated magnitude of the prediction errors.

V. Conclusions

We are far from satisfied with the current state of the assignment. Although the video player is working for every type of chroma subsampling, due to time constraints, the only method of encoding that is implemented is the intra-frame. We hope to be able to upgrade it to a hybrid encoder (intra-frame + inter-frame) in the future, as well as developing a lossy option.

References

[1] David Abrahams Beman Dawes. Boost C++ Libraries. http://www.boost.org/. [Accessed: 14 October 2017].

[2] M.j. Weinberger, G. Seroussi, and G. Sapiro. The loco-i lossless image compression algorithm: principles and standardization into jpeg-ls. *IEEE Transactions on Image Processing*, 9(8):1309–1324, 2000.