Automated Replay Detection and Multi-Stream Synchronization

Kevin Mekul, BSc
Alpen-Adria-Universität Klagenfurt, Austria
kmekul@edu.aau.at

ABSTRACT

This paper introduces an algorithm for automated replay detection in CS:GO tournaments. Firstly, it uses a content-based approach for replay identification in the commentator stream and then the metadata to synchronize them to the player streams. The provided metadata for events cannot be used for detecting events in the streams, because the timestamps of these events are very inaccurate and differ up to 40 seconds. Furthermore, the algorithm uses a content-based approach along with the synchronization data to detect the equivalent scenes in the matching player stream.

1 INTRODUCTION

E-sports are competitions which are only held virtually via video games. These sports are becoming increasingly popular nowadays. One of these games is Counter-Strike: Global Offensive (CS:GO), which is a multiplayer first-person shooter game. The CS:GO World Championship tournament was placed in Katowice in 2018. Each game is played between two teams of five players. Such events produce an enormous amount of multimedia data and metadata. During the tournament, the multimedia data of each game is composed of the commentator stream and ten player streams. It is not possible to process the material manually and therefore Mediaeval 2019 defined one task: *GameStory: The 2019 Video Game Analytics Challenge* [2]. The first part of this task is to find replays in the commentator stream and the second one to synchronize the streams and detect the replay scenes at the original player stream. A replay is a predefined scene of a player stream, which starts and ends with a logo in the commentator stream. Figure 1 shows this logo. The final goal is to run the implemented algorithm with the test dataset (video material of 03-03-2018) and submit up to three solutions. The processing of multimedia data was developed in Python in combination with OpenCV [1] and the metadata preparation was implemented in Java. The detection approach is a content-based variant and the algorithm can be divided into three steps, but each step uses the interim result of the step before:

- Replay detection in the commentator stream
- Processing the metadata
- Scene detection in the player streams

This subdivision is necessary, because the first step needs a lot of time, because the replay detection algorithm runs through the whole commentator stream and analyzes every frame.

2 REPLAY DETECTION

The tournament the GameStory challenge focuses on ran its course over three days and there is video material of each gaming day, which includes the commentator stream and the player streams.

MediaEval'19, 27-29 October 2019, Sophia Antipolis, France

Figure 1: The replay logo

Every player stream shows the perspective of a specific player. The commentator stream includes live scenes, replays of interesting events, statistics and much more. This video has a frame rate that differs from the player streams. Therefore, the algorithm adjusts the frame rate to 59 fps. This correction enables an acceptable basis for the following detection. According to detection speed, I resized the width to 450 pixels. In addition, the system converts color frames to gray scale images for further analysis.

A replay contains a specific scene with a pair of logo transitions and a sweeping off effect at the beginning and the end [4]. This shot is detected and defined only once, and the algorithm can detect the starting and endpoint throughout the whole video.

2.1 Detecting replay start

The replay start point detection is always identical. In contrast, there are five variants of the replay end.

The detection of a starting point is a full frame comparison based on the structural similarity index, which OpenCV provides. If the score is higher or equal than the threshold of 0.99, the algorithm found the start logo. The next step in this process is adding a predefined number to the just found frame number, to get the first frame after the logo transition. After this step, the algorithm starts at the adapted frame number to check four regions of interest (ROIs).

The first ROI is the name of the current shown player. The second one is the small Intel-Logo at the right side of the frame. If a replay is running, this logo will not be displayed. The third ROI analyses the current health, which is a number from 100 down to 0. The algorithm uses this ROI to detect the kill of the currently shown player. The last ROI analyzes the time.

While detecting the replay starting point it is possible, that no replay is found after the start replay logo. Because of this reason, the algorithm checks the first and second ROI. If that scene is no replay, the found start frame can be deleted and the search process for the start logo begins again. If the frame difference limit is reached, the algorithm will stop the search for the ending logo. If a correct start point for a replay is found, the algorithm will check the replay end scenarios.

2.2 Detecting replay end

The algorithm includes five end scenarios. The first three scenarios just stop the replay. The last two stop the current replay and start a new one. These five scenarios are:

- The standard stopping case for a replay is the logo transition at the end. This detection is the same as the start detection, only the adaptation of the frame number is different. The algorithm reduces the frame number with a predefined number.
- A replay will stop if the current shown player is killed. This happens, when the health of a player reaches zero. The algorithm uses the L1-norm of the normalized reference and the ROI values. If the norm drops below the threshold value, the current shown player will be killed, and the replay will stop at this frame.
- A hard shot, every pixel in the frame is black, stops the replay too. The black frame occurs after a normal frame and after the black one, a normal frame is shown again but from another scene. The algorithm detects this case with the OpenCV method absdiff and stops the replay. The next replay logo is not marked as a starting point, because it is the end of the prematurely stopped replay.
- The algorithm for detecting a player change uses the first ROI and the structural similarity index. The found frame is the last of the current replay and the next one is the beginning frame of the new starting replay.
- The system identifies a time jump based on the time ROI and compares the current time with the time from the 13th frame before.

The results of replay detection are the frame numbers for begin and end of a replay and the shown player. The metadata processing uses these interim results.

3 METADATA PROCESSING

The metadata processing uses the result from the detecting replay and generates a list with the *roundbegin* and *next_round_begin* frame numbers for the matching player stream. The Java program reads the *replay_begin*, *replay_end* and *replay_source* values from the commentator stream results and looks up the suitable match and round. After that, the algorithm uses the matching match and round to find the correct *round_begin* frame number of the player stream. This process uses the sync-points, but some rounds have no frame numbers. For that reason, it searches the *metadata.csv* for the specific day. This metadata is not usable because it is very inaccurate. As a result of using the *metadata.csv* the working time for the next step is longer. Fortunately, this *metadata.csv* was never used, because all detected replays are in rounds with sync-points.

4 SOURCE DETECTION

The replay source detection is the last step of the algorithm and is implemented with Python and OpenCV. This step uses the *round_begin* numbers to start the detection.

The algorithm compares two ROIs. First, it checks the time of the found replay starting frame of the commentator stream with the current time of the player stream until it fits. If the times match, the middle region of the same frame is checked until the suitable frame is found. The algorithm does the same with the replay and frame of the commentator stream. The detection stops if the matching start and end frame in the player stream is detected or the next round begins. The time ROI has one disadvantage: the program must resize the time of the commentator stream in order to compare it with the time ROI of the player stream, because the time ROI has different size in the commentator and the player streams.

The result of this step includes the replays of the commentator stream and the begin and end frame number of the accurate player stream.

Table 1: Result: test dataset

jc >=	precision	recall	F1	avg. overlap
0.50	0.9744	0.8636	0.9157	0.7458
0.75	0.9487	0.8409	0.8916	0.7595

5 RESULTS AND DISCUSSION

The approach I have chosen was mainly based on content-based methods. Only the generation of the synchronization part uses another approach. A Python script and a groundtruth file for the training dataset was available to check and verify the algorithm.

The next step was to check the algorithm with the test dataset. All parameters and configurations are the same as in the implementation step with the training dataset. The described approach detects 39 replays for the test dataset and the possible matching source scenes for that. The organization group checked this with a Python script. Table 1 shows the result.

This task is evaluated in 2 steps. Both, the found replays in the commentator stream and the found scenes in the player stream were checked. For the first step, the used methods are precision, recall and the balanced F1-score. For the second step an average overlap of the found sequence was computed. One detected replay is labeled with result 'None', because the video stream files of P7 and P10 shows in match 1 and round 1 to 4 the same player on that day.

ACKNOWLEDGMENTS

Special thanks to the organization group especially Mathias Lux and thanks to Michael Wutti for the provision of the synchronization points [3].

REFERENCES

[1] Joseph Howse. 2013. *OpenCV computer vision with python.* Packt Publishing Ltd.

[2] Mathias Lux, Michael Riegler, Duc-Tien Dang-Nguyen, Pirker Johanna, Martin Potthast, and Pål Halvorsen. 2019. GameStory Task at MediaEval 2019.. In *Proceedings of MediaEvel 2019.*

[3] Michael Wutti. 2018. Automated Killstreak Extraction in CS: GO Tournaments.. In *Proceedings of MediaEvel 2018.*

[4] Zhao Zhao, Shuqiang Jiang, Qingming Huang, and Guangyu Zhu. 2006. Highlight summarization in sports video based on replay detection. In *2006 IEEE international conference on multimedia and expo.* IEEE, 1613–1616.