REAL-TIME ENGINE SOUND GENERATOR BASED ON ANALYSIS OF VIDEO AND RECORDED SAMPLES

Marcin Skoczylas

Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: Generating engine sound samples is a broad topic known for decades, mostly because of high usage of such algorithms in driving car simulations, especially in games. These algorithms differ from very simple looped short sound players that change the frequency of prerecorded samples to sophisticated algorithms that model the engine sound based on some defined characteristics. The latter are computationally extensive and can't be used in mobile environment (such as smartphones). In this paper author presents own approach to use visual analysis techniques to prepare a database of multiple recorded sound samples and a mixer that can replay these sounds in proper order to mimic an engine sound in real-time.

Keywords: engine sound generator, recorded samples, mixer, visual anaylysis, Hough Transform

1. Introduction

Generating automotive engine sound samples is a broad topic known for decades, mostly because of high usage of such algorithms in games, especially driving car simulations, also professional ones such as presented in [14]. Engine sounds generator is very important for the feeling of speed during the simulation and is inseparable unit of the whole experience, giving constant impression of speed. To accurately model the sound of a motor vehicle, in direct response to the interactivity of the simulation, there are many challenges in order to represent it as realistic as possible. These algorithms differ from very simple looped short sound players that change the frequency of prerecorded samples to sophisticated algorithms that model the engine sound based on some defined characteristics. First approaches to simulate car engine sounds were done back in '80s using sine waves with eventually distortion added. The first racing vehicle sounds were simulated using Revolutions Per Minute (RPM) motor parameter and simple wave synthesis, so that the increase of RPM increased tone and motor sounds frequency. Even at this basic level, driving speed was communicated to the user and though these algorithms sounded very simple but were enough to attract. One of very popular algorithms to represent engine sound is to use simple, looped sounds with playback frequency correlated to the RPM value. The original engine sound is recorded, played in a loop and then its frequency is changed depending on target RPM of the engine. Other algorithms use procedural engine models to properly create dynamics in response to the player interaction, for example using FFT analysis and granular synthesis, distortion and effects such as delay or flange. These sophisticated algorithms model the sound based on some defined characteristics of the engine. The latter are computationally extensive and can't be used in mobile environment (such as smart phones).

In this paper author presents own approach to use visual analysis techniques to prepare a database of multiple recorded sound samples and a mixer that can replay these sounds in proper order to mimic an engine sound in real-time.

2. Related work

Modeling of engine sound is not very popular research topic. Most existing solutions are done commercially for the purpose of generating sounds for games and the idea behind algorithms is not publicly shared. However, there are few publications that are worth noting here. In [4] authors modeled and synthesized engine sounds using a deterministic-stochastic signal decomposition approach, the deterministic component was extracted using a FFT method and it was subtracted out from the original signal and then the stochastic component was modeled and synthesized using a new multipulse excited time-series modeling technique. The technique gives very good results, however is quite computationally extensive, thus can't be implemented in mobile smartphones. Very often the sound generation is connected with implementation of a physics engine, as presented recently in [15] or [13]. Authors created a framework that allows virtual object contact sounds to be synthesised in real time, eventually adding a possibility to create car engine sounds. A very promising work is presented in [9], where authors presented deep analysis of engine sounds and some ideas are also base for this publication. Other approaches also exist to generate engine sounds in real time, recently very popular became hardware boxes from Sonory Engine Sound Synthesis [1]. These boxes can be hooked up to in-car stereo radio system to replay artificial sounds of V8 engine based on RPM readings from internal computer. Although these are implemented in hardware, still the sound is simulated and synthesized thus it is not easily possible to change the characteristics to generate different engine sounds.

Engine sound analysis is much more popular topic in research, and mainly concerns engine fault detection as for example in [10], a mobile application that analyzes the engine sound and detects engine faults using discrete wavelet transform.

In this paper author presents own approach to use visual analysis techniques to prepare a database of multiple recorded sound samples and a mixer that can replay these sounds in proper order to mimic an engine sound in real-time.

3. Engine sound generator method

In this paragraph idea of the method to generate arbitrary engine sounds is presented. To avoid computationally extensive engine sound synthesis and obtain realistic effect, taking into account that the algorithm should be able to generate natural sounding vehicle engine audio of multiple different engines and it should be relative easy to switch and obtain sounds of other cars, author decided to create an algorithm that creates and automatically organizes a bank of short sample sounds that will be used for mixing by the engine sound player in real-time. In overall, to correlate engine sounds with RPM readings one can use a plug to the car's computer to obtain current RPM readings. Unfortunately, not all available cars easily support sending a feedback of the current RPM to a PC, especially old cars that do not have on-board computers. Thus to avoid this problem and allow engineers to record and model sounds of very old cars, a video analysis algorithm was created and current RPM is obtained from a video frame.

In overall, this method contains a step that involves sound and video analysis of recordings of the original sound of the engine. The setup consists of a video camera that records readings of the RPM dial and in addition microphone that is attached to the car body to record sounds. Sounds are recorded together with video of the RPM dial on site, eventually operator drives a car with a load attached. These recordings are then marked and transferred to a PC that performs video analysis. Sounds and video frames are extracted and analyzed using the algorithm described below to create a bank of organized short audio samples marked with the RPM readings.

3.1 Video analysis and extracting RPM readings

To allow users to record sounds of cars with significant load, recordings must be done within a car that is in motion. Very often the mounted camera creates shaky videos, especially when car drives on uneven terrain. Thus before to obtain a RPM reading from one video frame, first that video needs to be stabilized.

Many different video stabilization techniques exist. Main purpose of this step is to reduce in-between frames motion. Author decided to create his own approach to video stabilization based on detection of features in images and so-called keypoints detectors and descriptors. First a marker is defined manually, this could be a vehicle logo or some other element which is visible on all frames and that does not overlap the RPM dial. This marker image is used as a reference point to stabilize the video. From this marker image keypoints are detected and descriptors are calculated.

Image feature descriptors are becoming a standard in current state of the art of image recognition algorithms. For this study, author selected most common and popular feature detectors: Scale-Invariant Feature Transform (SIFT [12]), Speeded Up Robust Features (SURF [5]) and also recently presented Binary Robust Invariant Scalable Keypoints (BRISK [11]). Results of the accuracy of marker image detection are presented in section 4.

In a new video frame a marker image is detected. First, keypoints are found using the same feature detector, these keypoints form a set $K_c = \{p_1, p_2, ...\}$ and are considered as candidates for keypoints that correspond to the marker image. For all keypoints in the set K_c feature descriptors D_c are calculated, so that each element from set K_c corresponds to one descriptor from set D_c. A nearest neighbour kNN search is performed on detected keypoints in a new frame and marker keypoints with[1] $K = 2$. Found pairs are filtered to find good matches using technique described in [7]: first, the minimum distance (*min*) is found from all matches, and then all distances that are bigger than $2 \cdot min$ are discarded. If the amount of keypoints in a set containing found matches is less than 4 (thus, at least four corners), then the marker is not detected and that frame is skipped: stabilization is not performed, the marker image is not visible. In other case the marker is detected and a homography is found using a RANSAC [8] algorithm using pairs of keypoint matches and then perspective matrix transformation of vectors is performed. If the transformed polygon is not convex, then the marker is considered not detected.

When a position of the marker on a new video frame is known, then a relative position to the original position of the marker can be calculated. The whole video frame is then shifted in the opposite direction to overlay a marker from new frame on a marker in the previous (original) frame and video stabilization is continued for next frames.

[1] The k=2 in kNN is suggested by J. Beis and D. Lowe in their Best-Bin-First (BBF) algorithm [6]

3.2 Detection of angle of the RPM pointer

When video is stabilized a next step of the algorithm is detection of the RPM pointer and then its angle. Knowing the angle one can calculate also the RPM reading. All video frame images are processed to obtain number of RPMs connected with a video frame using following algorithm steps:

1. Image is converted to a gray scale color space.
2. Binarisation filter is applied with threshold calculated using Mean Iterative Selection [2]. During each iteration the average brightness T_B is determined for all pixels below the estimated threshold, and similarly also the average brightness T_W of all pixels above value of the estimated threshold. The new value of the estimated threshold value is calculated as the average of the two values T_B and T_W. The general formula for calculating the estimated value of the threshold for the histogram h:

$$T_k = \frac{\sum_{i=0}^{T_{k-1}} i * h[i]}{2 \sum_{i=0}^{T_{k-1}} h[i]} + \frac{\sum_{i=T_{k-1}+1}^{N} i * h[i]}{2 \sum_{i=T_{k-1}+1}^{N} h[i]} \tag{1}$$

   The moment of stopping the algorithm is the condition:

$$(T_B = T_W) \vee (T_{k-1} = T_k) \tag{2}$$

3. Boundaries of the dial are detected using a sweep algorithm and pixels that reside inside a circle of the RPM dial are extracted and considered for further analysis.
4. A Hough Transform is calculated. From the resulting Hough Transform matrix the best representative of the angle (θ) and offset of the most visible line is chosen. When these representatives are multiple then values are averaged.
5. RPM pointer angle is detected using data from the previous step. It is possible that angle will be not correct. The Hough Transform result can detect angle from opposite part of the circle, and in such cases that angle needs to be corrected. First a RPM dial is divided into 4 equal parts. Then, amount of pixels that reside in each quarter is calculated and from that it is known in which quarter the pointer is located. Then simple correction calculations are performed, for example, if pointer is detected in left-down quarter and θ is higher than $220°$then $\theta = |180° - \theta|$, etc.
6. Finally, a RPM angle value is linearly scaled to reflect RPM values range, thus for example $180°$ becomes 3000 RPMs and so on.

RPM pointer readings calculated from the angle are stored for all video frames. For each frame as a result from this step a RPM reading is stored together with short sound sample which has a duration of one frame: $1/(frames\ per\ second)$. These readings will be organized into bulk groups of samples. The algorithm is explained in following paragraph.

Example of the RPM pointer detection is presented in Figure 1, and Hough Transform matrix used to RPM pointer angle detection is visible in Figure 2.

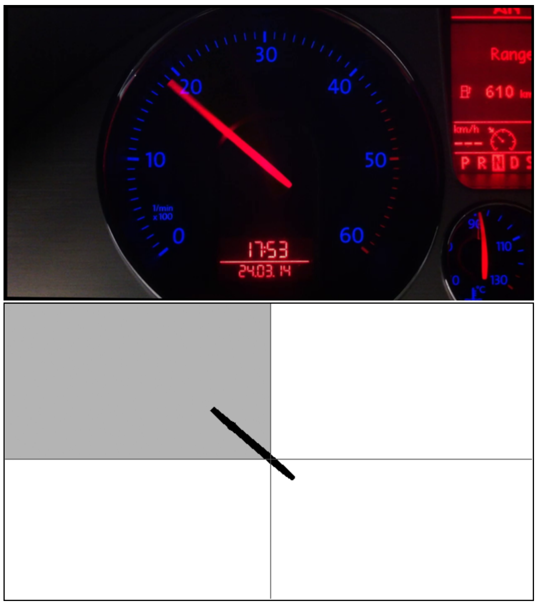

Fig. 1. Example of RPM pointer detection steps. Original video frame image, segmented pointer and divided dial with selected quarter highlighted.

3.3 Samples database creation

In previous paragraph all video frames were analyzed and RPM readings were stored. Next step organizes sound samples into three groups: constant RPM, accelerate and decelerate groups of samples. For this purpose vector of all RPM values is analyzed. First the RPM signal is smoothed using gaussian smooth filter and then for each frame discrete derivative is calculated. Knowing derivatives then groups of samples that consist of acceleration, deceleration or constant RPM values are found, however

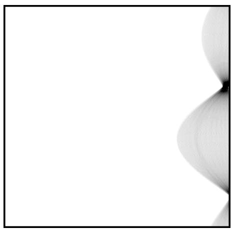

Fig. 2. Hough Transform matrix calculated on segmented pointer example from Figure 1.

Table 1. Example temporary data structure for storage of detected RPMs together with video frames. From this structure three groups of samples are created.

Frame #	RPM	State	Start Frame #	End Frame #	Min RPM	Max RPM
1	750	Constant	1	3	750	770
2	770	Constant	1	3	750	770
3	760	Constant	1	3	750	770
4	800	Accelerate	4	6	760	900
5	850	Accelerate	4	6	760	900
6	900	Accelerate	4	6	760	900
7	850	Decelerate	7	9	900	750
8	800	Decelerate	7	9	900	750
9	750	Decelerate	7	9	900	750

to avoid erroneous state changes with too rapid differences, frames are analyzed in clusters of 3 frames. Thus, to confirm detection of a state change from constant to acceleration, then three consecutive frames have to show the RPM acceleration. In addition an information in which frame this group started and in which frame that state ended is stored, as well as RPM for that group of samples. All groups of samples are kept in three separate buckets: constant RPMs, accelerations and decelerations. See as an example Table 1 where the concept is shown. Please note that RPM for a frame starts as a previous value and finishes as current frame's RPM (for example see frame #4). This example stores links to three groups of samples detected, one constant, one accelerate and one decelerate group. Start, end frames and start, end RPMs are stored with groups and dataset is created.

3.4 Method to select group of samples during playback

The real-time player of the engine sounds is playing recorded samples based on information obtained after analysis from the previous sections. The input parameters to the player are target RPM (R_{target}) and the engine load. The engine load parameter just selects proper bank of previously analyzed samples. R_{target} is a RPM value to which the engine should aim and select consecutive sound samples.

The algorithm selects samples to be played based on dependencies of current RPM ($R_{current}$) and target RPM (R_{target}). Knowing RPM of the sound sample that was previously played the algorithm selects new samples to be played as follows:

1. If the R_{target} is equal $R_{current}$ then select a random group of samples from 'Constant' samples bucket that has RPM equal to the R_{target}.
2. If the R_{target} is higher than $R_{current}$ then select a random group of samples from 'Accelerates' samples bucket that has minimum RPM equal or below the $R_{current}$ and the maximum RPM equal or higher than the R_{target}. If the $R_{current}$ is not equal starting RPM of this group of samples then scroll the sample to frame that matches $R_{current}$.
3. If the R_{target} is less than $R_{current}$ then select a random group of samples from 'Decelerates' sample bucket that has minimum RPM equal or higher than the $R_{current}$ and maximum RPM equal or below the R_{target}. If the $R_{current}$ is not equal starting RPM of this group of samples then scroll the sample to frame that matches $R_{current}$.

The above algorithm is run every sample step. Note, that if a current group of samples that is played resides in 'Accelerates' bucket and maximum RPM from the group is higher than R_{target}, then that group is replaced by a group of samples from the 'Constant' bucket when $R_{current}$ reaches R_{target}. Analogously, if a current group of samples that is played resides in 'Decelerates' bucket and minimum RPM in a group is lower than R_{target}, then that group is replaced by a group of samples from the 'Constant' bucket when $R_{current}$ reaches R_{target}. That group replacement is achieved by the first step of the algorithm.

To quickly select proper group of samples a Red-Black tree is used as a storage structure and search algorithm. In addition number of uses of a particular group of samples is recorded. When a group of samples is selected to be played then the number of usages is increased. Algorithm selects new group of samples based on the amount of previous usages, thus it favors selection of a group that was not played before than replaying the same again. This approach avoids repeatable loops of the same group of samples.

Furthermore, a classical volume (amplitude) ramping method [3] is used, so each sample group to be played is overlapped during mixing with previous to avoid clicks and noise in a place of cut (see Figure 3).

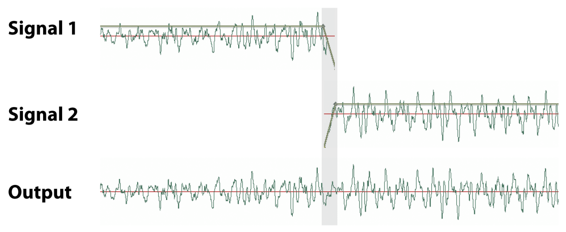

Fig. 3. Ramp-mixing of two groups of samples (signals). Green horizontal lines over the signals show output volume. The Output signal does not have clicks when two signals are joined.

4. Results

4.1 Comparison of marker detection algorithm

First results of marker detection algorithm from section 3.1 for the purpose of video stabilization will be presented. Image marker was extracted from first video frame and other video frames were processed. Resolution of the marker was 240×160 pixels. Normally video recording parameters can change, so to mimic this video frame images were changed: scale, noise, rotation, blur and lightness filters were applied and detection results gathered. A *ratio* parameter was obtained in such way: first from the original marker image keypoint descriptors were calculated, then video frames were changed and on these amended images marker was detected using algorithm from section 3.1. The *ratio* is a number of properly matched keypoints in a new image (that are positioned inside marker area) divided by the original number of keypoints from the marker. A ratio of 1.0 means that all the keypoints from the marker image were found properly in the amended image. Results of this experiment are presented in Table 2. It is clearly seen that SURF algorithm is performing best video stabilization, surprisingly weak result of BRISK algorithm can be explained by low resolution of the marker image.

4.2 Engine sound generator

It is not easy to create a comparatory review of the algorithm presented in this paper. It is possible however to create a spectrogram of sounds generated by real vehicle engine and compare it with results generated by the algorithm presented. These spectrograms should not be equal, as real engine sound is not generating exactly the same sound every time and also it depends on the RPM throttle, so frequencies will vary over time. But visualization of such spectrograms gives information about common

Image filter	Ratio			Time		
	SURF	SIFT	BRISK	SURF	SIFT	BRISK
LIGHTNESS	96.9	94.0	60.7	27	28	2
NOISE	99.1	98.7	95.7	75	50	5
SCALE	90.5	87.8	40.4	133	123	9
ROTATE	98.4	96.6	66.7	42	49	5
BLUR	97.0	96.8	18.5	39	38	2

Table 2. Video-stabilization marker detection accuracy and algorithm run time. The higher ratio value, the better. The lower time value, the better.

frequencies to some extent. Example spectrograms that show differences in real vehicle sound and generated ones are visible in Figure 4, so they can be compared. Author performed also a subjective study on the quality of the playback on a limited number of people, however this topic needs to be studied further.

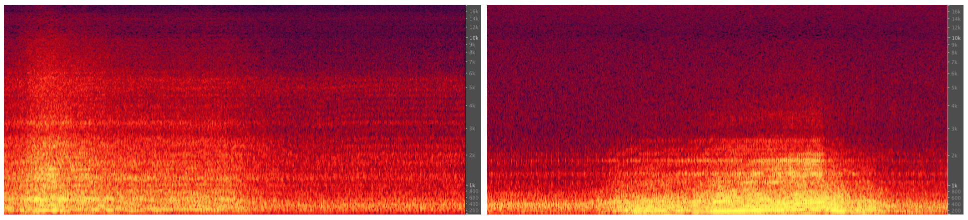

Fig. 4. Spectrogram of real car engine sound (left) and generated by the algorithm (right).

5. Conclusions

Modeling of engine sound is not very popular research topic. Most existing solutions were implemented commercially for the purpose of generating sounds for games and the idea behind algorithms is not publicly shared. Author created his own method to generate vehicle engine sounds that uses visual analysis techniques to prepare a database of multiple recorded sound samples and a mixer that can replay these sounds in proper order to mimic an engine sound in real-time. The solution can be used in a limited environment, such as for example on mobile smartphones. The algorithm generates audio that can successfully mimic sounds of vehicle engine and it can be used for the purpose of driving car simulation or computer games, especially in a limited environment.

References

[1] http://www.sonory.org/engine-sound-processor.html.

[2] http://www.olympusmicro.com.

[3] http://music.columbia.edu/pipermail/music-dsp/2006-April/065216.html.

[4] S.A. Amman and M. Das. An efficient technique for modeling and synthesis of automotive engine sounds. *Industrial Electronics, IEEE Transactions on*, 48(1):225–234, Feb 2001.

[5] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[6] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. *In Conference on Computer Vision and Pattern Recognition, Puerto Rico*, pages 1000 – 1006, 1997.

[7] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[8] M. Fischler and C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[9] J. Jaglaa and J. Maillard. Sample-based engine noise synthesis using an enhanced pitch-synchronous overlap-and-add method. *J Acoust Soc Am.*, 132(5):3098–108, Nov 2012.

[10] Y. Karunakar, A. Kuwadekar, and K. Al Begain. A mobile based application for detecting fault by sound analysis in car engines using triangular window and wavelet transform. In *Computational Intelligence and Communication Networks (CICN), 2010 International Conference on*, pages 523–528, Nov 2010.

[11] S. Leutenegger, M. Chli, and R. Siegwart. BRISK: Binary Robust invariant scalable keypoints. *Computer Vision, IEEE International Conference on*, 0:2548–2555, 2011.

[12] D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[13] E. Mullan. Driving sound synthesis from a physics engine. In *Games Innovations Conference, 2009. ICE-GIC 2009. International IEEE Consumer Electronics Society's*, pages 1–9, Aug 2009.

[14] Tao Ni, Dingxuan Zhao, and Hongyan Zhang. Realistic vehicle driving simulator with dynamic terrain deformation. In *Mechatronics and Automation, 2009. ICMA 2009. International Conference on*, pages 4795–4800, Aug 2009.

[15] Ting Wei and Hao Zheng. Sound effect of physical engine in game design. In *Information Technology, Computer Engineering and Management Sciences (ICM), 2011 International Conference on*, volume 3, pages 148–151, Sept 2011.

Marcin Skoczylas

GENERATOR DŹWIĘKU SILNIKA W CZASIE RZECZYWISTYM NA PODSTAWIE ANALIZY WIDEO I ZAREJESTROWANYCH PRÓBEK

Streszczenie: Generowanie dźwięku silnika jest szerokim tematem znanym od dziesięcioleci, głównie z powodu zastosowania takich algorytmów we wszelakiego rodzaju symulacjach jazdy samochodem, a w szczególności grach komputerowych. Algorytmy te stosują różne podejścia, m.in. od bardzo prostych odtwarzaczy zapętlonych dźwięków, które zmieniają częstotliwość nagranych uprzednio próbek do zaawansowanych algorytmów modelowania dźwięku silnika na podstawie określonych cech charakterystyki silnika. Algorytmy te są obliczeniowo skomplikowane i nie mogą być stosowane w urządzeniach przenośnych (takich jak np. smartfony) w czasie rzeczywistym. W tym artykule autor przedstawia wĹ‚asne podejście do korzystania z technik analizy wizualnej aby automatycznie przygotować bazę wielu nagranych krótkich próbek dźwiękowych oraz miksera, który odtwarza uporządkowane dźwięki w odpowiedniej kolejności, tak aby naśladować sterowalny dźwięk silnika w czasie rzeczywistym.

Słowa kluczowe: generator dźwięku silnika, nagrane próbki, mikser, analiza wizualna, Hough Transform

Artykuł zrealizowano w ramach pracy badawczej MB/WI/3/2012.