Sound and Music for Interactive Games: Car Audio System Project Report

Jim McGowan
Leeds Metropolitan University
MSc Sound and Music for Interactive Games
j.mcgowan4782@student.leedsmet.ac.uk
jim@bleepsandpops.com

INTRODUCTION

This project is an attempt to create an audio system to recreate sounds for use in a racing game. The audio system was created using Audio Units [1], the audio signal processing system available on the Mac OS X and iOS operating systems. However, the concepts and structure of the audio system are implementation-agnostic and could be recreated in any other modular audio processing platform, for example Cycling 74's Max or Pure Data (PD).

The audio system is presented in an game-like application to drive the system and demonstrate it in a gaming context. However, although this application was created by including the audio system code with the code for the other components, the audio system remains a distinct entity and could also reside in a separate process or application, receiving physics data via a system such as Open Sound Control.

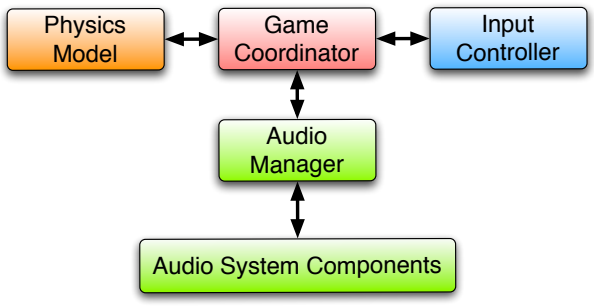
1.AIMS

This project aimed to create an car audio system that was generic in the sense that with the appropriate input the system could believably recreate the sound of any particular car that might be desired within a racing game. Additionally, the system should not present a significant processing load to the host system, allowing it to be used effectively in resource-restricted game contexts.

2.GAME IMPLEMENTATION

The audio system is presented in a game-like application in order to demonstrate its capabilities. The complete application is structured as show in figure 1 below:

Figure 1. Game Application Structure.



The 'game' is managed by the Game Coordinator object, which presents the car selection UI, starts and stops the game, and maintains the main game loop. The Input Controller reads input from the keyboard, the Car object represents the physics model of the car, and the audio manager presents the interface to the audio system, which it maintains.

On each pulse of the game clock the Game Coordinator reads the accelerator, brake and steering input values from the Input Controller and passes these to the Car physics model. It then triggers the Car's periodic state update function, and reads the updated Car state values (speed, RPM, etc) and passes these to the Audio Manager which updates the audio system to suit.

2.1Specifying Cars

As mentioned above, the system aims to be generic enough to believably recreate any car if the correct input is presented. To this end, cars are specified in an XML file that details the car's performance specifications and the identifies of the corresponding audio samples. The performance data is used by the physics model to accurately recreate the performance of the car and the audio files are used by the Audio Manager to popular the audio system. The values contained with in a car XML file are described in Appendix 1.

3.AUDIO SYSTEM

The audio system is based on recorded looping samples, and is similar in nature to the common crossfading sample approach used in some games [2, 3, 4, 5]. Other approaches were investigated during the course of the project, including granular and synthetic approaches. However, such systems require extremely precise recording and modeling of each specific car and engine to be recreated, which presents a significant barrier to entry. A looping sample system is simpler to provide audio material for and, as this project will hopefully show, can produce believably realistic audio output.

The audio system is modular, consisting of subsystems for engine, exhaust, turbo, tyres, road sound and a master mix section. Each of these is discussed in detail below. A diagram of the complete system is shown in Appendix 2. The subsystems are contained and controlled by an Audio Manager system that presents an Object-Oriented API (Application Programming Interface) to the host application or game. This manager creates an instance of the audio system when required, receives physics data from the host application/game, and passes the relevant data to the relevant subsystem. The manager also deals with high level operations, such loading the required audio samples indicated in a car's XML specification and setting overall mix presents.

Maintaining the audio system as a collection of subsystems brings several benefits. It allows for easy expansion should there be a desire for additional audio elements be included in the system. Individual subsystems can have their internal structure modified

without impact on other areas. Separation of processing into logical areas also makes code reading and system maintenance simpler.

3.1 Engine and Exhaust Subsystems

The Engine subsystem is structured as shown in figure 2 below:

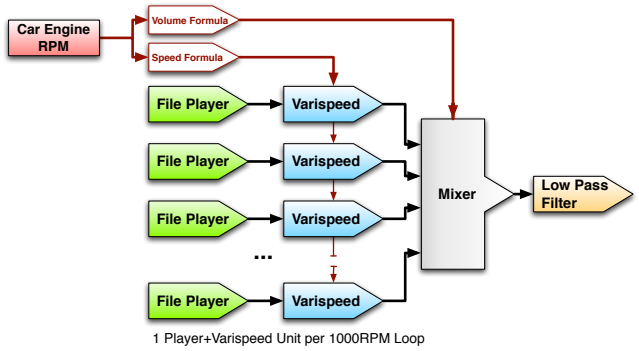


Figure 2. Engine Audio Subsystem.

One file player unit and varispeed unit are created for each engine loop sample provided for the specified car. These are connected to a mixer which manages the relative levels of each sample. The mixer in turn is connected to a low pass filter unit, which can be used to simulate the car bonnet being open or closed, or the listening position being within the engine bay.

The engine subsystem requires the car's current engine speed (RPM) as input. This is used to calculate the required volume and speed of each loop sample. The speed of each sample is calculated by dividing the engine RPM by the sample's native RPM, which gives a linear increase from 0 speed at 0 RPM to full speed at the sample's native RPM, and linearly onwards. The formula for this can been seen at line 221 in the source file EngineExhaustSoundSystem.m

The volume of each sample (in a scale of 0 for silent to 1 for unity gain) increases linearly from 0RPM to 1 at the sample's native RPM, remains at 1 from the native RPM to 1.3x the native RPM, after which it decays logarithmically. The graph below shows the volumes for a five sample system. Engine RPM is mapped over the x-axis and sample volume over the y-axis.

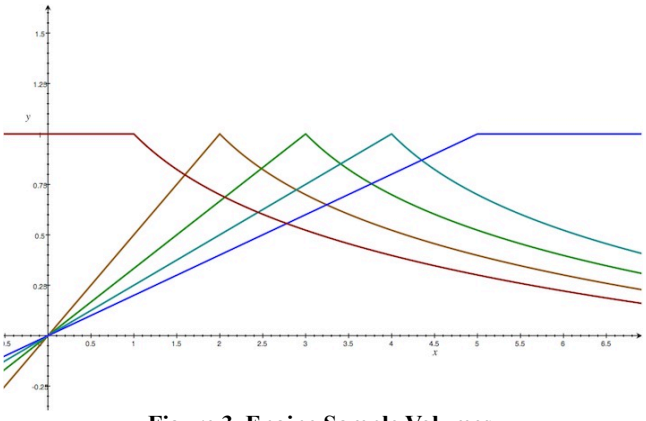


Figure 3. Engine Sample Volumes.

As the graph shows, two special cases exist: the first sample should be at full volume constantly until the engine RPM reaches its native RPM; and the last sample should play at full volume from its native RPM upwards.

The code for the volume formula can be see at line 229 in the source file EngineExhaustSoundSystem.m.

At the beginning of this project the engine and exhaust systems made use of the more 'standard' crossfading approach, where each sample is at full volume at its native RPM, and an equal power crossfade is applied to fade the sample out when the engine RPM reaches the native of the next higher sample, which has faded in over the same period. This is shown in the graph below:

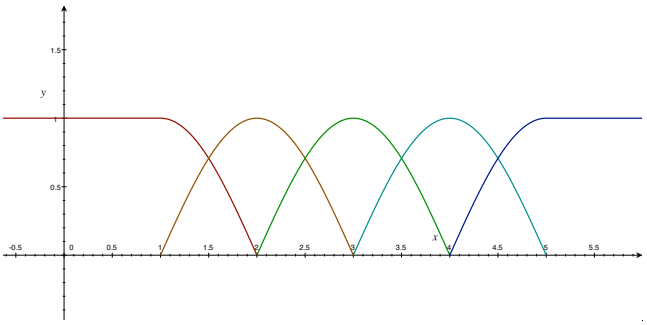


Figure 4. Crossfading System Sample Volumes.

However, I found that in order to maintain a consistent sound as the car revs up and down, realism in the sound had to be sacrificed. At lower speeds a car engine produces more noticeable high frequency content, from tappets, rods and other small items. The main engine 'drone' from the combustion, pistons and crankshaft is at a relatively low intensity. As the engine speed increases the intensity of the main drone increases sharply, and the higher frequency elements increase further in pitch and are less present in the overall sound. Using the between-sample crossfade system the different spectral content of the samples at different RPMs are great enough (at 1000 RPM intervals at least) that the transitions between samples are very obvious to the listener and do not sound natural. In order to get a natural sound as the car increases and decreases RPM over its full range, the individual samples had to be heavily processed to smooth the transitions. This had the effect of losing some of the distinct characteristics of each engine, reducing it to simpler drone. Experimentation showed that the linear-up/logarithmic-down technique used gave a much more smooth and natural sound as the engine speed increases and decreases.

The exhaust audio subsystem is exactly the same as the engine subsystem, but does not make use of the low pass filter at the mixer output.

3.2 Turbo Subsystem

The turbo subsystem is structured as shown in figure 5 below:

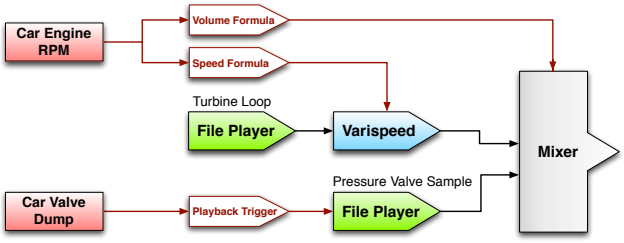


Figure 5. Turbo Audio Subsystem.

A file player unit and varispeed unit are used to playback the turbine loop. A file player unit is used to playback the pressure release sample. These elements are connected to a mixer.

The turbo subsystem requires notifications of the car's pressure release valve triggering and the car's engine speed as input. The pressure release notification simply triggers the playback of the non-looping valve sample. The engine speed is used to determine the volume and speed of the turbine loop.

The turbine volume curve is given by the formula:

$$Volume = \frac{\sqrt[3]{\frac{engineRPM}{1000}}}{2}$$

This gives a rapid increase in volume at low speed, leveling out slightly at higher speeds. The code for this can be seen at line 193 in the source code file TurboSoundSystem.m

The turbine speed increases exponentially with engine speed, and is given by the formula:

$$Speed = \frac{\left(\frac{engineRPM}{1000}\right)^{1.5}}{3}$$

The code for this can be seen at line 190 in the source code file TurboSoundSystem.m

3.3 Tyres Subsystem

Based on advice from practicing game audio designers [5, 6], the audio system has individual elements for wheel spin (where the wheels' forward speed is faster than the car's forward speed) and skidding (where the tyres are moving in a direction other than perpendicular to their axle).The tyres subsystem deals with these sounds and is structured as shown in figure 6 below:

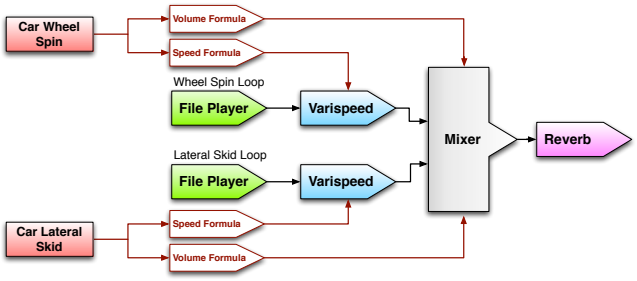


Figure 6. Tyre Audio Subsystem.

A file player/varispeed pair is used for the wheel spin loop and another for the skid loop. These elements are connected to a mixer, which is in turn connected to a reverb unit.

The tyre subsystem requires car wheel spin and lateral skid data as input. These values have a range of 0 for no skid/spin to 1 for complete loss of tractions. These are used to calculate speed and volume values for the spin and skid sounds. The volume of the spin sound is the car's wheel spin value doubled and constrained between 0 and 1. This allows the sound fade up to full volume as the spin value reaches 0.5, which gives a natural sound. The code for this can be seen at line 276 in the source code file TyreSoundSystem.m.

By the same reasoning the skid sound volume is 3x the car's skid value, constrained between 0 and 1. To reduce repetitiveness the start point of the skid loop is randomized each time a skid begins Therefore a quick fade in is applied over the volume values when a skid begins to prevent any clicking sounds should playback begin at a point of non-zero power in the audio file. The code for this can be seen at line 288 in the source code file TyreSoundSystem.m.

To mimic the increased intensity of wheel spin and skid sounds at high speeds the playback speed of each loop is varied. The spin sound playback speed is 1.5x the car's spin value and the skid sound playback speed is 1.5x the car's skid value. The code for these formulas can be seen at lines 282 and 327 respectively in the source code file TyreSoundSystem.m.

As traction data from the physics system can drop off abruptly, the reverb unit is used to help provide a natural decay to the tyre sounds. The values for the reverb unit can be seen at line 230 in the source code file TyreSoundSystem.m.

3.4 Road Subsystem

The road subsystem reproduces the sound of the car's tyres traveling over the road surface. It is structured as shown in figure 7 below:

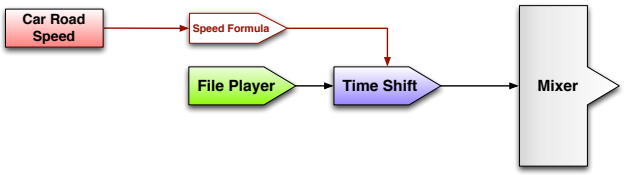


Figure 7. Road Audio Subsystem.

A file player unit plays the road noise loop, and is connected to a time shift unit, which provides a time-compression/expansion (speed change without pitch change) effect.

This subsystem requires the car's road speed as input. The volume of the road sound fades up linearly from 0 at 0KPH to 1 at 30KPH to give a natural sound. The code for this formula can be seen at line 187 in the source code file RoadSoundSystem.m.

The playback speed increases with road speed based on the formula:

$$PlaybackSpeed = \left(\frac{CarSpeed}{75}\right) + 0.6$$

The output of this formula is constrained to between 0.6 and 1.9. This formula gives a natural increase, hitting normal speed at 75KPH and continuing to increase thereafter. The code for this formula can be seen at line 178 in the source code file RoadSoundSystem.m.

3.5 Master Section Subsystem

The master section subsystem provides a mixer with the output from each of the other subsystems as inputs. This is used to adjust the relative levels of each subsystem when moving listening positions around the car. The master system also contains and low pass filter unit on the output of the mixer, which can be used when the listening position is moved to within the car.

From an implementation point of view, the master section subsystem also maintains the Audio Units Graph, which is the required context within which the Audio Units reside. Therefore this subsystem is also responsible for starting and stopping audio rendering and is the point at which hardware usage can be monitored.

3.6Preparing Audio Samples

For the example 'game' application I prepared 3 cars: a 2010 Mercedes Benz E250 CDI, a 2010 Audi Q7 Quattro Diesel and a 2000 Volkswagen Beetle.

I recorded engine and exhaust audio at 1000 RPM increments for each. The engine speed of each was monitored using only each car's dashboard tachometer. Therefore the recorded sounds at each speed were not at exactly 1000RPM intervals and it was therefore necessary to make small adjustments to the speed of the recordings. The 4000RPM engine recording as a fixed reference, chosen only as 4000 divides easily. This sample was slowed to 25%, 50%, 75% speeds and increased to 120% speeds to give reference samples at 1000, 2000, 3000 and 5000 RPM. The other recordings had their speed adjusted so that they were in tune with the reference samples. The 4000RPM exhaust samples were tuned to match their 4000RPM engine counterpart, and the process was repeated for the exhaust recordings.

Other than fine tuning the speed and editing to loopable segments, no other processing was applied to the engine and exhaust sounds, in order for the audio system to produce as realistic an output as possible.

Figure 8. Recording VW Beetle Engine Sounds.

Various recordings were made of the Volkswagen Beetle traveling over tarmac at different speeds with its engine off to create material for the road noise loop. These were edited to give a loopable, steady-speed sound, in the range of around 40KPH. Each of the three cars shares the same road loop.

I did not have the resources to record a turbo charger in isolation (i.e., not attached to a running engine), which was unfortunate as the audio system requires clean turbo sounds. After listening to descriptions and sample recordings of turbo sounds, as presented in a lecture on racing game audio [2], I discovered the fundamental sonic elements of a turbo charger's turbine are a large amount of white noise and a very high pitched fan sound, very similar to the sound of a hand held hair dryer. Therefore, I made recordings of a hair dryer and manipulated them to recreate a close approximation to reference sound presented in the lecture. The turbo's pressure release valve sound is that of the rapid release of compressed air, therefore I was able to recreate this by editing compressed air recordings.

Additionally, I did not have the resources to record the three cars skidding and wheel spinning. For these sounds, I edited recordings from the Freesound.org project of a Chrysler skidding on a track and of a Volvo's wheels spinning. Individual spin and skid elements were extracted from these recordings and used to create loopable samples. Each of the three cars share these same samples.

4.PHYSICS MODEL

As seen above, the audio system requires realistic and accurate physics data in order to accurately reproduce car sounds. The physics model created for the project is crude but, with exceptions discussed below, is able to provide usable data for the audio system.

The car physics model has performance attributes that are loaded from the car's XML specification: max torque, engine idle RPM, engine max RPM, max braking force, auto shift up and down RPMs, aerodynamics, weight, tyre circumference, final drive ratio, gear ratios and gear change time. These attributes are immutable and are used in calculations to update the car's state.

The model has three adjustable parameters for input: accelerator pedal position, brake pedal position, and steering wheel position. These values are updated by the Game Coordinator on each each pulse of the game clock, based in the values received by the Input Controller.

On each each pulse of the game clock the Game Coordinator triggers the car physics model's update function, which updates the car's state based on the input parameters and its performance characteristics. The process of update function is as follows:

1. *Calculate force on road from driven wheels*
2. *Calculate braking force*
3. *Calculate air resistance (drag) force*
4. *Calculate friction force*
5. *Sum these forces*
6. *Calculate the acceleration from the summed forces*
7. *Apply the acceleration to the car's velocity vector*
8. *Calculate traction values by comparing the velocity angle and magnitude to the wheel speed and steering angle*
9. *Update road speed and engine RPM based on velocity and traction*
10. *Change gear if required.*

The car physics model has the following state values that can change on each update:

- •Engine RPM
- •CarAngle
- •Traction
- •Lateral Skid
- •Road Speed
- •Wheel Spin
- •Current Gear
- •Overall Velocity Vector.

After triggering the model's update function the Game Coordinator reads these updated values and passes the relevant data to the Audio Manager, which in turn passes the data to the subsystem inputs.

5.SYSTEM PERFORMANCE

5.1Processing & Ram Performance

CPU load of the audio processing system is very low. Measurements were taken on an Apple MacBook Pro with an Intel Core 2 Duo processor running at 2.93GHz. When a car is at idle the audio processing system uses between 0.16% and 0.2% of the overall available processing power. When a car is running at speed, the audio system uses between 0.23% and 0.25%. Should this system be used in a race involving eight cars, the car audio processing load would remain below 2%.

The example Mercedes car has the largest set of sample files. The samples in the example 'game' application are uncompressed 48kHz, 24 bit AIFF files, with a combined data size of 11MB. However, with tighter editing of the loops, sample rate reduction on the lower frequency samples and compression, this data size can be reduced to around 500KB

5.2Audio Performance

5.2.1Engine Audio

The accompanying audio file 'Audio System Acceleration' is a recording of the output of the audio system when the player is accelerating and decelerating in the Mercedes car. The file 'Real Car Acceleration' is a recording of a Mustang Mystic Cobra during a race (recorded from within the car). Comparing the two recordings, the pitch changes over the course of acceleration are very similar. However, in the real car recording a sudden short increase then decrease in engine pitch can be heard immediately after completing a gear change. This detail is not present in the sound from the audio system.

Having investigated this detail whilst driving a real car, it was revealed that this is caused by a driver's foot beginning to depress on the accelerator pedal whilst gradually releasing the clutch. The engine is able to increase speed whilst disconnected from the drive system, then this speed is reduced as the engine is reconnected with the clutch release and encounters the friction of the drive system.

Clutch engagement and disengagement in the physics model in the example application is entirely binary - its state is either fully engaged or disengaged - and changes between the two states are therefore instantaneous. With a input data from a more detailed physics system that modeled the analogue nature of the clutch pedal, I believe the audio system would accurately recreate these engine pitch details when completing a gear change.

The accompanying audio file 'NFS Game Acceleration' is a recording of car audio from Electronic Arts' game *Need for Speed: Shift*. Comparison with the audio system output shows again that the engine pitch curves are very similar. In the game recording the post-gear change engine RPM detail heard in the real car recording are not present, which is an interesting omission. Unfortunately it is not possible to determine from the game alone whether this is a result of the game's physics system or a sound design decision.

In the game recording there is an initial burst of high revs as the car pulls away from rest, caused by wheel spin reducing traction so that less resistance is carried up the drive system to the engine. (although the actual tyre spin sound is not clearly heard in this recording, it can be heard at other points during gameplay). This is a much more believable sound that the very short high rev burst and wheel spin screech heard on the audio system recording.

There are two factors causing this less believable sound. The first is that the wheel spin calculations within the physics model are too crude and do not produce particularly useful data. Attempts were made to introduce compensative adjustments to the model, but these simply produced an excess of spin, making control of the car very difficult.

The second cause may be that the tyre subsystem in the audio system may be to tightly coupled to the physics data. Discussions with the audio designer for the Need for Speed game series [5] revealed that unless wheel and tyre physics data is exceptionally accurate, tyre audio system needs to be de-coupled from the physics system and a more audio-aesthetic approach taken to create believable sounds.

5.2.2Skid Audio

The accompanying audio files 'Audio System Skid', 'Real Car Skid' and 'NFS Game Skid' are recordings of skid sounds from the audio system, the Mustang Mystic Cobra racing and the *Need For Speed: Shift* game respectively.

The audio systems skid sounds compare quite well with those from the real car and the game, though there are some details that could be improved. In the real car skids the sounds of individual tyres can be discerned, which isn't the case for either the audio system or the game. This could be mimicked in the audio system with multiple instances of the tyre subsystem, one representing each wheel, though this would require precise per-wheel physics data as an input.

The variations in pitch are slightly more pronounced in the real car and game recordings than in the audio system sound. The speed formula within the tyres subsystem could be adjusted to create greater variations.

6.FURTHER DEVELOPMENT

As discussed above, the limitations of the physics system used within the example 'game' application prevent the full potential of the audio system from being explored. It would be of great benefit to couple the audio system to a more precise and detailed physics model that could provide more realistic input. The Open Racing Car Simulator (TORCS) [7], a racing game engine and research platform, and VAMOS, "an automotive simulation framework with an emphasis on thorough physical modeling" [8] are both open source C++ projects that could possibly be used as sources of better physics data.

The control interface for the example 'game' application (use of the computer's arrow keys) is very limited due the binary on/off nature of each button. Receiving input data from an input device with analogue controls, such as an analogue game pad, could provide much more realistic input data to the system.

With better input and physics data, the audio system could be further developed in the knowledge that it is platform-agnostic - that implementation details within the audio system are not compensating for poor input. This would allow the audio system to become truly independent and portable for use in different games.

The road sound subsystem currently only allows for one road surface. This could be expanded to included different road

surface as might be found around a race track (tarmac, gravel, grass). An input from the host game system would be required to indicate which surface sound to use.

When utilizing the audio system within a racing game, multiple instances of the system would be required, one for each car in a race, and the output of each instance would need to be localized in 3D game space to match the location of the corresponding car. There are several ways in which the 3D sound locations could be managed, depending on the implementation of the game system. however by continuing to make use of Audio Units, the 3D sound processing could be achieved very simply and all the audio processing would be contained within the audio system.

The Macintosh and iOS operating systems include an Audio Unit entitled 3DMixer which can "can mix audio from several different sources and then localize the sound in 3D space" [9]. If each instance of the car audio system remains within the enclosure of a single object like the Audio Manager in the example 'game' application, that object could maintain a 3DMixer unit and update its state based on 3D positional information from the game engine. The output of the 3D mixer can be configured for stereo, quadrophonic or 5.0 rendering.

To demonstrate how this might be achieved figures 9 and 10 show the audio structure as used in the example 'game' application, and how this would be extended for multiple cars in 3D space.

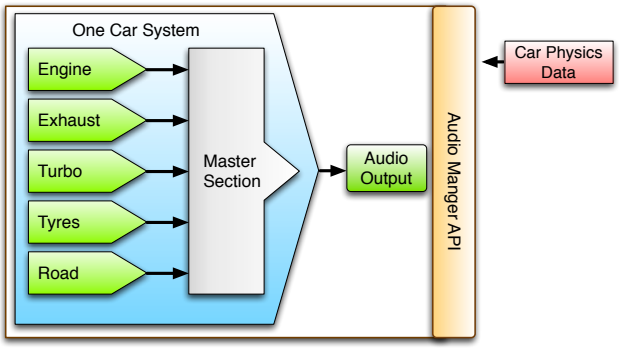


Figure 9. Existing System.

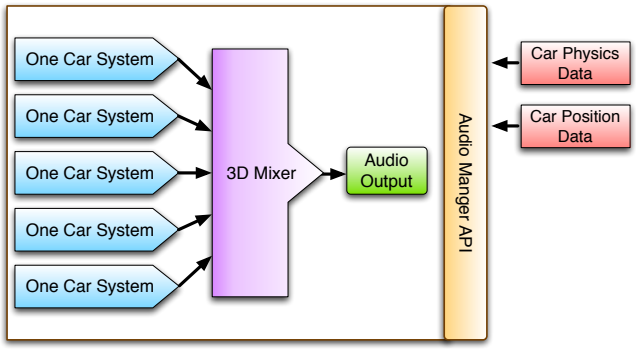


Figure 10. System Expanded With Multiple Cars in 3D.

Of course the simplicity of this 3D extension is only applicable whilst the system is implemented using Audio Units. Should the system be ported to another audio processing platform, the 3D implementation would differ.

7.CONCLUSIONS

Overall the project meets its aims with reasonable success. The generic-adaptable nature of the system is demonstrated by the different performance and audio characteristics of the three different cars included in the demonstration application. Overall the hardware resource requirements of the audio system are low. Finally, though some improvements may be made, the audio output compares favourably with recordings of real cars and car systems from existing games.

REFERENCES

[1] Apple, Inc. *Turning Up the Volume with Audio Units*. 2006. Retrieved 21 January 2011: http://developer.apple.com/library/mac/#technotes/tn2004/tn2112.html

[2] Caviezel, M. & Robinson, J. Advanced Audio Techniques for Racing Games: Creating Audio for Forza Motorsport2. *In: Game Developers Conference, 5-9 March 2007, San Francisco*. CMP Game Group 2007.

[3] Caviesel, M. Forza Motorsport 3: The Design, Process and Pipeline of a Car. *In: Game Developer's Conference Canada, 6-7th May 2010, Vancouver.* CMP Game Group, 2010.

[4] Munro, K. Field Report: High Octane Recording and Implementation of Engine Audio. *In: Game Audio Summit, GDC Austin, 15-16 September 2009, Austin, Texas*. CMP Game Group 2009.

[5] Deenen, C., *Car Audio System*. Personal Email to Jim McGowan. 4 January 2011.

[6] Kastbauer, D., *Implementing Car Tyre/Skid Sounds*. Personal Email to Jim McGowan. 28 December 2010.

[7] Wymann, B. & Espié, E. *Torcs*. Retrieved 21 January 2011: http://torcs.sourceforge.net/

[8] Varner, S. *Vamos Automotive Simulator*. Retrieved 21 January 2011: http://vamos.sourceforge.net/

[9] Apple, Inc. *Technical Note TN2112: Using the 3DMixer Audio Unit*. 2004. Retrieved 21 January 2011: http://developer.apple.com/library/mac/#technotes/tn2004/tn2112.html

FURTHER BIBLIOGRAPHY

Farnell, A. *Designing Sound*. MIT Press, 2010.

Finlay, D. Power and Torque Explained, *Car Keys*, 2002. Retrieved 30 November 2010. http://www.carkeys.co.uk/features/technical/power_and_torque_explained.aspx

Macro Monster, *Physics for Car Games*. 2003. Retrieved 29 November 2010. http://www.asawicki.info/Mirror/Car%20Physics%20for%20Games/Car%20Physics%20for%20Games.html

Zuvich, T, *Vehicle Dynamics for Racing Games*. Article of unknown origin.

ACKNOWLEDGEMENTS

The reference audio of the real car race is from the video *NASA MA Summer Breeze Sunday Summit Point Thunder Race 8-22-10* by Frank Corkran, available from http://www.vimeo.com/14410796

The spin and skid sounds in were created from the recordings *Chrysler LHS tire squeal 01 (04-25-2009), Chrysler LHS tire squeal 02 (04-25-2009), Chrysler LHS tire squeal 02 (04-25-2009),Chrysler LHS tire squeal 04 (04-25-2009)* and *AE0090 Volvo 740 GLE handbrake turn 01* created by user audible-edge available from the Freesound.org project.

Appendix 1: The Car Specification XML File

This table details the values found the in the car specification XML files that are used as input to the physics and audio systems.

Value Name	Value Unit or Type	Required or Optional	Description
Name	Text	Required	The name of the car
maxEngineRPM	In Revolutions per Minute (RPM)	Required	The maximum RPM of the car's engine
maxTorque	In Newton meters (Nm)	Required	The Maximum torque produced by the car's engine
maxBrakingForce	In Newtons (N)	Required	The combined maximum braking force from all the car's wheels
engineIdleRPM	In Revolutions per Minute (RPM)	Required	The car's engine idle speed
autoShiftUpRPM	In Revolutions per Minute (RPM)	Required	The engine speed at which the auto gear system will shift upwards
autoShiftDownRPM	In Revolutions per Minute (RPM)	Required	The engine speed at which the auto gear system will shift downwards
aCd	Unitless number	Required	The cars frontal area multiplied by its drag coefficient
weight	Kilograms (Kg)	Required	The car's weight
tyreCircumference	Millimeters (mm)	Required	The circumference of the car's tyres.
finalDriveRatio	Decimal number rated against 1	Required	The car's final drive (or axle) ratio
gears	Array of decimal number rated against 1	One array element required for each gear	The car's gear ratios in a sorted array with first gear as the first element. Reverse is not included.
gearShiftTime	In seconds	Required	The length of time taken to change gear
engineSounds	Array of filenames	At least 1 filename required	The engine sample names in an array. The first sample must be at 1000RPM. If more than 1 sample is listed, they must be at 1000RPM
exhaustSounds	Array of filenames	At least 1 filename required	The engine sample names in an array. The first sample must be at 1000RPM. If more than 1 sample is listed, they must be at 1000RPM.
turboSound	Filename	Optional	The name of the optional turbo turbine sample
turboDumpSound	Filename	Optional	The name of the optional turbo pressure release valve sample
skidLoopSound	Filename	Required	The name of the required skid loop sample
wheelSpinSound	Filename	Required	The name of the required wheel spin loop sample
roadSound	Filename	Required	The name of the required tyres-on-road-surface sound

Appendix 2: The Full Audio System

This chart outlines the complete audio system.