SIREN

Sound Generator for Vehicle Simulation

Authors
Anders Andersson, VTI
Anders Genell, VTI

www.vipsimulation.se
Preface

*Sound generator for driving simulators* is a collaborative project between Swedish National Road and Transport Research Institute (VTI), HiQ, Volvo Trucks, Volvo Car Corporation, Saab Automobile and Scania within the competence centre *Virtual Prototyping and Assessment by Simulation* (ViP).

The scope of the project was to develop a simulator sound engine to be used by ViP partners. The software should be generic, i.e. it should be able to use with little amount of effort, and able to work in conjunction with current hardware-in-the-loop (HIL) requirements.

This report comprises the final documentation of the project which has been financially supported by the competence centre ViP, i.e. by ViP partners and the Swedish Governmental Agency for Innovation Systems (Vinnova).

The software developed in the project is available at the ViP common software platform vipforge, [https://www.vipforge.se/](https://www.vipforge.se/). Access is granted to ViP partners by the vipforge administrator Björn Blissing at VTI (bjorn.blissing@vti.se).

Participants from VTI were Anders Genell (project manager) and Anders Andersson.

Participants from HiQ were Peter Paunonen and Linda Nordin.

Participants from Volvo Trucks were Per Nordqvist, Pontus Larsson and Nina Teodorsson.

Participants from Volvo cars were Eva Lahti, David Lennström and Fredrik Hagman.

Participant from Saab Automobile was Anders Sköld.

Participant from Scania was Johan Fagerlönn.

Gothenburg, April 2012

*Anders Genell*
Quality review

Peer review was performed on 12 January 2012 by Hans Habberstad, FOI. The authors, Anders Andersson and Anders Genell, have made alterations to the final manuscript of the report. The ViP Director Lena Nilsson examined and approved the report for publication on 12 November 2013.
Table of contents

Executive summary ......................................................................................................... 7
1  Introduction .................................................................................................................. 9
  1.1 Traditional means of vehicle sound simulation ................................................. 9
  1.2 The new approach ................................................................................................. 10
  1.3 Project scope ........................................................................................................ 11
  1.4 Aims ....................................................................................................................... 11
  1.5 Definitions ........................................................................................................... 12
2  Sound systems ............................................................................................................ 13
  2.1 Simulator sound systems ..................................................................................... 13
  2.2 Development environments ............................................................................... 17
  2.3 Sound Management ............................................................................................. 17
  2.4 Sound Generation ................................................................................................. 21
3  Results ....................................................................................................................... 27
  3.1 SIREN Functionality ............................................................................................ 27
  3.2 Computational performance ............................................................................... 29
  3.3 Audio performance ............................................................................................. 30
  3.4 User feedback ...................................................................................................... 31
4  Discussion ................................................................................................................ 32
  4.1 Experience from using SIREN ........................................................................... 32
  4.2 Experience from developing .............................................................................. 32
  4.3 Future development of SIREN .......................................................................... 34
5  Conclusions – What has been provided through SIREN? .................................. 36
References ...................................................................................................................... 37
<table>
<thead>
<tr>
<th>Abbreviations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALSA</td>
<td>Advanced Linux Sound Architecture</td>
</tr>
<tr>
<td>ALUT</td>
<td>OpenAL Utility Toolkit</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller Area Network</td>
</tr>
<tr>
<td>JACK</td>
<td>JACK Audio Connection Kit</td>
</tr>
<tr>
<td>LGPL</td>
<td>GNU Lesser General Public License</td>
</tr>
<tr>
<td>OpenAL</td>
<td>Open Audio Library</td>
</tr>
<tr>
<td>OSC</td>
<td>Open Sound Control</td>
</tr>
<tr>
<td>Qt</td>
<td>Qt ['kjuːt] Graphics Toolkit</td>
</tr>
<tr>
<td>Sim II</td>
<td>VTI Simulator II</td>
</tr>
<tr>
<td>Sim III</td>
<td>VTI Simulator III</td>
</tr>
<tr>
<td>Sim IV</td>
<td>VTI Simulator IV</td>
</tr>
<tr>
<td>SIREN</td>
<td>ViP Simulator Sound Renderer</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1: Simple sine tone (left) compared to complex recorded sound snippet (right) 10
Figure 2: VTI Simulator II ........................................................................................................ 13
Figure 3: VTI Simulator III .................................................................................................... 14
Figure 4: VTI Simulator IV .................................................................................................... 15
Figure 5: Volvo XC60 cabin surround sound system in VTI Sim IV ............................... 16
Figure 6: Volvo FH12 cabin surround sound system in VTI Sim IV ............................... 16
Figure 7: Class layout of the sound management in SIREN ............................................ 20
Figure 8: Engine order analysis using FFT of sound recorded in a Volvo FH12 truck
equipped with a Volvo iShift automatic gear box, performing a wide open throttle
(WOT) acceleration from stand still to 90 km/h. Engine orders 0.5, 1, 1.5, 2, 2.5 and 3
can be seen marked in blue ..................................................................................................... 23
Figure 9: Comparison between a sound measurement in a real truck and a measurement
of corresponding sound generated in the Sim IV truck cabin ........................................... 30
Figure 10: Comparison between a sound measurement in a real car and a measurement
of corresponding sound generated in the Sim IV car cabin .............................................. 31
List of Tables

Table 1: List of introduced dependencies in SIREN sound management with short descriptions.................................................................................................................................................. 18

Table 2: Sound functionality currently implemented in Sim IV through SIREN ........ 29
**SIREN – Sound Generation for Vehicle Simulation**

by Anders Andersson and Anders Genell  
Swedish National Road and Transport Research Institute (VTI)

**Executive summary**

The ViP Simulator Sound Renderer (SIREN) software has been created as a means to facilitate generation and playback of audio signals in driving simulators. Siren is a modular, scalable program with a plug-in based infrastructure. The included plug-ins offer sound file playback, sound stream playback and spatialization possibilities. Required additional functionality can be added by creating custom plug-ins. Siren by default relies on the OpenAL library for spatialization and on Csound for sound stream generation. Other spatialization and generation software can be used by replacing the corresponding API modules.

Siren is implemented in the new Simulator IV as well as in Simulator III at VTI and will also be implemented in Simulator II in the immediate future. Experimental implementations have been tested in the VTI Foerst simulator running solely under the Microsoft Windows operating system. Volvo Trucks has a trial version implemented in their simulator and has made some local customization. The current sound models implemented through Siren in the VTI simulators consist of real-time synthesis of sound based on measurements performed in real vehicles (car and truck) on the Volvo test track. The resulting sound has been validated through corresponding measurements performed inside the simulator cabins as well as through informal listening by experienced drivers.
1 Introduction

Simulating a driving environment usually includes an auditory feedback from the simulator system to the driver. Sound in the environment is comprised of, among other things, the own vehicle engine, the noise from the tires and from surrounding traffic. The audio signal gives feedback both about the vehicle state and the surrounding environment. Depending on driving task and the experience of the driver, the auditory feedback can be of great importance for the handling and performance (Genell, 2008). Particular cases, such as low frequency (infra) sound (Sandberg, 1983) or combinations of sound and vibrations (Ihs, Andersson, Kircher, & Bolling, 2010) have shown to affect driver behaviour in earlier simulator studies. One of the key requirements of a simulated environment is to achieve what is generally called “presence”, i.e. the feeling of “being there” (Larsson, Västfjäll, & Kleiner, 2004). A way to achieve high presence is to provide cues that make the user an active part of the simulated environment, e.g. through self-motion (Väljamäe, 2007). For auditory information this means providing a realistic sound field allowing our hearing to perceive the expected spatial nature of the surroundings (Blauert, 1997). The more realistic the sound is the better the experience will be for the driver and, perhaps more importantly, the more reliable the inference from experiments.

1.1 Traditional means of vehicle sound simulation

In simulators as well as in computer games it is common to represent the sound from a vehicle engine by snippets of recorded sound played in a loop, where the snippet recorded at a speed of revolution close to the current virtual engine speed of revolution is continuously chosen. While this is a perfectly valid method for representing sound inside a vehicle, there are some limitations that may become problematic.

First, obtaining the sound snippets themselves is a challenge in itself since the recorded sound is relatively complex. A pure sine tone, which is as simple as a sound can be, can be divided into snippets by identifying where the recorded sound wave is at its minima, so that the onset and offset of sound coincides with the sound wave boundaries. As soon as the sound is becoming complex such natural boundaries are difficult if not impossible to identify (cf. Figure 1). This means that the onset and offset of the sound will be audible as a repeating click. To hide such clicks the sound snippet needs to be slowly faded in and out, and to avoid silent periods while the sound is being faded another copy of the same sound is started just before the previous has ended, thus cross-fading between the two.

Second, if the properties of the simulated vehicle are changed, a corresponding change of engine sound requires a cumbersome repetition of the process of identifying new sound snippets. As there are roughly 100 snippets needed to cover the entire revolution speed range, and roughly 10 levels of engine load for each speed of revolution, there are 1000 snippets to create.
The previous sound systems in the VTI driving simulators Sim II and Sim III employ the recorded sound snippet scheme, and when planning for the construction of the new Sim IV an early decision was taken to adopt a more versatile model. Also, the ability to easily add effect sounds, for e.g. driver support systems, and to allow for three dimensional sound were high on the wish list. All these requirements have now resulted in the development of a new sound rendering system, SIREN.

1.2 The new approach
The ViP Simulator Sound Renderer (SIREN) is a novel approach to the sound simulation situation. When developing or refining a simulator environment to enhance presence there are usually several properties that can be improved. However, the auditory experience seldom has top priority. This mostly leads to a situation where there is a large potential for improvement, but where the time to commit is lacking. In the work with SIREN this problem has been addressed by creating a framework that is easily extended using pre-defined modules referred to as plug-ins. In a previous ViP study at VTI a sound engine based on the Open Audio Library (OpenAL) was used (Fagerlönn, Andersson, & Liljedahl, 2011), and was found to present a good foundation for a spatial “sound engine”, as it is platform independent and (in the “OpenAL Soft” version) open source software, i.e. free to use under the LGPL license. Concepts used in the previous OpenAL-based sound engine have been adopted in the development of a new sound rendering system, SIREN.
manager, focusing on the ability to manage both event-based sound file playback and real-time generated sound streams.

Key design features of SIREN are stability, portability and modularity.

SIREN is freely available for all partners in ViP with the hope that it will be used and further developed by the partners involved.

1.3 Project scope

The project scope was to develop a simulator sound engine to be used by partners in ViP. The software should be generic in the sense that any partner should be able to use it with little amount of effort. It should also be able to work in conjunction with current hardware-in-the-loop (HIL) requirements.

1.4 Aims

The aims of the project were that the developed sound engine software should:

1. manage all sound in the simulator environment
2. be able to generate sound from vehicle models in real-time, and
3. be tested in a simulator environment to get reliable validation.

1.4.1 Sound management

The aim for development of the sound management was that it should be able to present all sound needed for the current simulator environments, and easily adapt to meet future needs. Specific focus was given to enabling spatial sound reproduction and moving sound sources.

1.4.2 Sound generation

The aim for the sound generation was to replace the previous design, i.e. application of interpolated loop points to recorded sound, with real-time sound synthesis, in order to allow for flexible variation of sound properties according to arising needs such as new vehicle types.

1.4.3 Simulator hardware dependencies

The aim for hardware compatibility was that the software should be modular, scalable and platform independent in order to be able to run on a large variety of existing hardware at the different ViP partners. Microsoft Windows and GNU/Linux were chosen as the main platforms to be supported. By using appropriate cross platform software libraries, the need of direct access to any sound card through (potentially proprietary) drivers and firmware was avoided.

1.4.4 Software dependencies

The aim for library dependencies was that while such dependencies should be kept to a minimum, reinventing the wheel by way of implementing functionality that is readily available should also be avoided.
In an earlier project (Fagerlönn, Andersson, & Liljedahl, 2011) the Open Audio Library (OpenAL) was found to be a powerful tool for handling spatialization and mixing of multiple sound sources. As it is also platform independent, has built-in support for a large variety of soundcards and is free to use under the GNU/GPL license, it was chosen as the basis for the sound manager development.

The Qt Graphics Toolkit is a platform independent environment which was chosen, in conjunction with the Boost collection of portable C++ source libraries, to facilitate cross platform support for properties that are platform specific, such as handling of User Datagram Protocol (UDP) sockets. To conform to, among other things the design of most parts of the VTI simulator software, it was decided that run-time configuration files should be written in the Extensible Markup Language (XML), and thus, a light-weight XML library was also included.

Csound is a platform independent, open source, real-time sound synthesis software and was chosen as the basis for sound generation.

1.5 Definitions

The following definitions have been used:

**Sound system**
The term sound system refers to the complete setup. This includes both hardware and software.

**Sound engine**
The term sound engine refers to the software used for the sound system.

**Sound management**
The term sound management refers to the part of the sound engine responsible for playback and spatialization of sounds in the simulated environment.

**Sound generator**
The term sound generator refers to the part of the sound engine responsible for creating sound streams in real-time, which are sent to the sound management system in the sound engine.
2 Sound systems

Over the course of the project a lot of different sound systems have been involved. Listed here are most of the sound systems used when developing and testing SIREN. Also included are some sound systems which were taken into account when designing SIREN, although these systems may not yet have undergone any extensive testing.

2.1 Simulator sound systems

Different simulators are built in different ways and it is unusual that they have common hardware. Since there are differences in the hardware there usually needs to be some software adjustment. A current overview of the driving simulators at VTI can be found on the web (VTI, 2013).

2.1.1 VTI simulator II

Figure 2: VTI Simulator II

The VTI Simulator II, Sim II (Figure 2), is a moving base driving simulator designed and constructed by VTI. The motion system consists of a linear motion, outer roll and pitch motion and a shake table. The visual system consists of six SXRD projectors which each has a resolution of 1920x1080 pixels presenting a 105 degrees field of view. Normally heavy vehicle driving is simulated using a Scania T112H cabin mounted in the simulator.

VTI simulator II sound system

Currently there are two sound systems installed in Sim II:

LjudprogramLastbil sound system

LjuddprogramLastbil is a sound engine developed by VTI. The sound engine runs on a Microsoft Windows computer with an external sound card that supports up to eight channels. The sound card is connected to two 5-channel Rotel amplifiers that powers speakers in the truck cabin. The speakers are placed at the existing mount points inside the cabin, i.e. in the doors and in the dashboard, apart from the subwoofer which is installed in the truck cabin wall.
The software is written specifically for the installed sound card (Terratec EWS88MT) in a Microsoft Windows environment and it is used to simulate the own vehicle sounds such as engine sound and tire/road noise.

**VTISoundEngine sound system**

To meet the demand for better spatialization of sound, an additional sound system was installed. Six Anthony Gallo Nucleus Micro speakers were placed at appropriate angles around the driver’s seat such that a 6.0 surround sound system was achieved. The speakers are connected to a Harman/Kardon AVR-255 amplifier placed next to the cabin, together with an Apple Mac Mini computer and an M-Audio ProFire 610 external sound card.

The software used, VTISoundEngine, was written in the Objective-C programming language for the Apple OSX platform and uses OpenAL. For further description of this sound system see (Fagerlönn, Andersson, & Liljedahl, 2011).

### 2.1.2 VTI simulator III

**Figure 3: VTI Simulator III**

The VTI simulator III, Sim III (Figure 3), is a moving base driving simulator where the design of the motion platform has many similarities with Sim II, but which allows smoother, large linear stroke, lateral motion as well as yaw motion. The visual system consists of 3 DLP projectors presenting a 120 degrees field of view. The main focus has been on passenger vehicles but the possibility to change to a truck cabin has been utilized, as well as the possibility to rotate the platform 90 degrees for large translational movement.

**VTI simulator III sound system**

The Sim III sound system is similar to the LjudprogramLastbil system installed in Sim II. Depending on which cabin is mounted on the motion platform it consists of the LjudprogramPersonbil or the LjudprogramLastbil sound engine running on a Microsoft Windows computer, with the same Terratec EWS88MT type sound card as in Sim II. The sound card is connected to an 8-channel Rotel amplifier placed on the motion platform that drives speakers placed at existing mount points inside the cabin, and a
subwoofer in the cabin wall. An additional amplifier drives one Buttkicker™ vibro-tactile actuator connected to the cabin floor, when using the Saab car cabin.

2.1.3 VTI simulator IV

Figure 4: VTI Simulator IV

VTI simulator IV, Sim IV (Figure 4), is a moving base simulator which was taken into operation in 2011, thus it is the newest of the simulator facilities at VTI. Examples of features of the simulator are a relatively simple procedure for changing cabins, a 6-DOF motion platform with linear and lateral stroke of 5 meters, and a visual system with 9 projectors providing a 210 degrees forward field of view. Currently there are two cabins available: a Volvo XC60 car cabin and a Volvo FH truck cabin.

VTI simulator IV sound system

The Sim IV sound system consists of a rack placed behind the cabin on the motion platform containing a dedicated sound computer with an RME HDSP 9652 internal sound card providing up to 26-channels, two Marian ADCON 8-channel ADAT D/A converters, one 8-channel C10:8X Lab Gruppen power amplifier which delivers 125W per channel, and one 4-channel C48:4X Lab Gruppen power amplifier which delivers 1200W per channel. The computer is a Hewlett-Packard DL120 rack mounted server with a 1.8 GHz dual core CPU and 2 Gb of RAM. It is currently running the Xubuntu 10.04 GNU/Linux operating system with a real-time kernel. All audio signals are provided to the mounted cabin through two multi-pol modular connectors.

A design requirement when installing speakers in the cabins was that the existing cabin speaker mounts were to be used where possible. In some cases extra loudspeakers have been fitted in additional positions to provide a proper surround sound field. As a result the speaker setup differs substantially between cabins.
**XC60 cabin**

In the Volvo XC60 cabin there are six speakers mounted in the existing speaker mounts (marked red in Figure 5, four extra speakers installed in additional positions on the B-pillars and behind the driver’s seat (small blue circles in Figure 5), and one subwoofer installed in the cabin back wall (large blue circle in Figure 5). Two Buttkicker™ actuators are used to produce vibrations in the cabin floor.

![Figure 5: Volvo XC60 cabin surround sound system in VTI Sim IV](image)

**FH cabin**

In the Volvo FH truck cabin there are four speakers mounted in the existing speaker mounts in the cabin doors and in the overhead compartment over the windshield (marked red in Figure 6), two extra speakers installed in additional positions behind the driver’s seat (small blue circles in Figure 6), and one subwoofer placed behind the passenger seat (large blue circle in Figure 6). Two Buttkicker™ actuators are used to produce vibrations in the cabin floor.

![Figure 6: Volvo FH12 cabin surround sound system in VTI Sim IV](image)
2.1.4 VTI simulator First sound system

VTI simulator First is a small, commercial (Foerst GmbH), moving base simulator with two degrees of freedom. The front view is presented on three screens. This simulator was delivered with pre-configured software, but VTI has also implemented its own simulator software providing the possibility to use either simulator environment.

The sound system has the same hardware setup for both environments, but the sound engine differs between using the Foerst simulation software or a stripped down version of the LjudprogramPersonbil/LjudprogramLastbil. The entire system runs on a Microsoft Windows computer with a soundcard connected to a Logitech Z-5500 speaker system in a 5.1 surround system configuration.

2.2 Development environments

Over the period of development there have been a couple of different setups used. The main development platforms have been a reasonably modern desktop computer and a low-end FiTPC2 computer.

2.2.1 HP Z400 desktop computer

The HP Z400 computer has an Intel Xeon W3520 2.67 GHz processor and 6 Gb of RAM. It has a Creative Sound Blaster X-Fi sound card installed which is connected to a Logitech Z-5500 5.1 surround speaker system. The main operating system is the 64 bit Microsoft Windows 7, but a 32 bit Ubuntu GNU/Linux operating system virtualized using VM VirtualBox has also been used in the development.

2.2.2 The FiTPC2 sound system

The FiTPC2 is a very small (115x100x25 mm) fan-less computer with very low energy consumption. The CPU is a 1.6 GHz Intel Atom Z530 that uses less than 7 W of power at full load, and the computer has 1 Gb of RAM installed. The operating system used is 32 bit Ubuntu Server 10.04 LTS using a real-time kernel. For audio output, a SoundBlaster X-Fi 5.1 USB external sound card connects to two pairs of JUSTer AC-691N desktop speakers in a 4.0 surround setup.

2.3 Sound Management

2.3.1 Key design features

The main features of the Sound Management were decided to be Stability, Portability and Modularity.

Stability

Stability is considered a very important aspect of SIREN and ideally SIREN should be up and running without interruption for several months at a time. During this time the sound engine shall not crash and no strange and or unwanted noise should appear. In practice this means that when there are several ways to solve a problem, as there usually are, the one considered most stable should always be chosen.
Portability

As SIREN should be made able to run by different partners within ViP the feature of portability is almost forced. With portability it is meant that SIREN should run on many different systems, e.g. Windows and Linux, and be able to use several different sound cards both internal and external.

The aim is that a lot of different operating systems with different sound card setups should be supported but as a start Windows and Linux (Ubuntu) were chosen to be supported and sound cards available were used.

Modularity

As there are several libraries to use, and different partners within ViP might want to have the possibility to change libraries if they feel the need for it, modularity was chosen to be a key feature. This means that the source code should be modularized with well-defined interfaces so that different parts can be exchanged easily. The extra work still needed for changing a module to one using a different library is not considered a part of the SIREN project as such, but will instead be performed by the relevant ViP partner as needed.

2.3.2 Source code dependencies

When designing the sound engine with these key features, two different dependencies were introduced. These were Qt and OpenAL. With Qt the intent was to simplify portability while also providing possibilities of a graphical user interface. As OpenAL had been initially tested, see (Fagerlönn, Andersson, & Liljedahl, 2011), it was chosen for managing sound spatialization on different platforms with both different operating systems and different sound cards. OpenAL is a platform independent sound programming library designed for efficiency and for handling three dimensional sounds. Doppler effects and distance attenuation on sound sources, which can be read from a file or streamed or any other sound source in PCM format, are automatically handled in addition to position and motion of sound sources around the listener. OpenAL has been used in several large computer game productions such as Doom3, Bioshock and Race Driver: GRID among others.

Some extra dependencies were introduced to save time. These libraries were Boost, Alut and TinyXML. These mainly pre-fabricated applications of C++ programming of the OpenAL functions, and the code they represent, could be directly included in the SIREN code base to avoid these dependencies if needed. Short descriptions of the different libraries can be seen in Table 1.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boost</td>
<td>Free peer reviewed portable C++ library extending C++.</td>
</tr>
<tr>
<td>Qt</td>
<td>Framework for cross platform and user interface development.</td>
</tr>
<tr>
<td>OpenAL</td>
<td>Free cross platform sound library.</td>
</tr>
<tr>
<td>Alut</td>
<td>Library with higher level convenience functions for OpenAL.</td>
</tr>
<tr>
<td>TinyXML</td>
<td>Free, open source, small and simple xml reader.</td>
</tr>
</tbody>
</table>
2.3.3 Cross platform and cross ViP partner development

Development of the source code has mainly been done by VTI and HiQ. As Qt was chosen as a cross platform development environment, QtCreator was used on the Windows systems and qmake in combination with the GNU/Emacs text editor was used on Ubuntu/Linux systems. Qt uses a project specific (.pro) file where the user specifies which files should be compiled. It also specifies which libraries to use and where to find them for the different operating systems. In this file there is also a possibility to specify compilation flags so that the source code can be compiled differently depending on system, e.g. by automatically optimizing for a specific CPU family.

To distribute workload weekly developer meetings were held where it was decided what should be done next. Handling of the source code was in the beginning done by sending files by mail or FTP but was later changed by introduction of a subversion (SVN) version handling server located at VTI. This allowed for parallel and largely independent development between developer meetings, dramatically increasing efficiency. To synchronise the source code layout the VTI kernel style guide was largely employed although there is still some parts of the source code that needs to be formatted accordingly.

2.3.4 Source Code Layout

The sound management source code can be divided into four main areas which are:

- SoundController and SoundObjects
- Plug-in handling
- OpenAL interface
- Utilities

There is also a user interface which can be used to interact with the sound management for testing and troubleshooting. A complete view of the source code class layout can be seen in Figure 7.
The main classes for controlling the sound management are the SoundController and \textit{SoundObjects}.

The SoundController is responsible for maintaining a soft real-time loop. A soft real-time system is a timer system that controls if a task missed a deadline, and if a deadline was missed the program warns the user while still continuing the execution. During this loop the SoundController updates used plug-ins, updates sound object information from the plug-ins, and updates streaming sound objects. At start-up the SoundController reads an xml file where it initializes parts of the sound management.

**SoundController and SoundObjects**

The main classes for controlling the sound management are the SoundController and \textit{SoundObjects}.

The SoundController is responsible for maintaining a soft real-time loop. A soft real-time system is a timer system that controls if a task missed a deadline, and if a deadline was missed the program warns the user while still continuing the execution. During this loop the SoundController updates used plug-ins, updates sound object information from the plug-ins, and updates streaming sound objects. At start-up the SoundController reads an xml file where it initializes parts of the sound management.
SoundObjects is a container class where different types of sound objects are stored and functions for adding, removing and updating sound objects are managed. Currently available are SoundObject and StreamSoundObject. SoundObjects are identified by a string when stored which is a design choice to easier identify sounds. SoundObjects can be created when needed during execution or pre-loaded from an xml configuration file at initialization.

Plug-in handling

Different simulators have different interfaces for how they handle information about objects in their simulator environment. This implies that there needs to be a possibility for partners to implement their input interface to for example network communication. This leads to the decision to have plug-in handling classes. A Plug-In is a pure virtual base class which the own developed plug-in inherits from. This base class have functionality for handling information and prepare the sounds used, e.g. receive positions and queue them to be sent to the SoundObjects so that the correct sound object is updated.

The handling also contains a creator base class, PlugInCreator, and a manager class, PlugInManager. The manager keeps track of every plug-in available and when a plug-in is specified in xml configuration the manager creates specified plug-in. Thus there is a possibility to have several different plug-ins available but only creating the ones needed.

OpenAL interface

A key design feature is modularity and due to this it was decided to have an interface to OpenAL. This interface resulted in the classes ApiToSoundLib and ApiToSoundStreamLib. These classes set up the OpenAL environment and contain all specific OpenAL code. The different APIs are called from the SoundObject or the StreamSoundObject classes.

Utilities

To provide functionality to the above mentioned classes there are utility classes. These are a timer class, a program log class, an xml reader/writer class, network communication classes and Open Sound Control, OSC, communication classes. OSC is a protocol for communication used in several sound applications.

2.3.5 Documentation

As a tool to document the sound management source code doxygen was used. Doxygen is a tool where you write the documentation in the source code and then doxygen generates the documentation. Classes that have not been documented are the inherited plug-in classes (the base classes are documented), the utility classes and the xml classes. Generated documentation can be viewed as a homepage.

2.4 Sound Generation

The sound generation is the part of the sound system responsible for providing real-time generation of streaming sound to the sound manager.
2.4.1 Software

Csound is an open source, cross platform, sound synthesis software capable of real-time generation of complex sounds. It has been under development for more than 20 years and is hence extensively tested and optimized, also on hardware with relatively limited resources such as the One Laptop Per Child\(^1\) system, where Csound is the default audio software. Csound is controlled by commands contained in a mark-up text file, using the suffix .csd for automatic recognition, where initialization, definitions and performance are handled. First there is an Options part, defining the relevant settings for the related instance of Csound. The settings are dependent on the sound card, and it is where the correct parameters need to be set in order for Csound to synchronize with the other parts of SIREN, especially with OpenAL which also depends on the properties of the sound card. Second, there is an Instrument part, defining how the actual sound should be created within an arbitrary number of user defined instruments.

Finally there is a Score part, where the sounds created in the previous part are played using numerical definitions of instrument, start time, duration and additional optionally added run-time variables. The Score part is useful for automated playback, but there is also the option to control instruments externally through e.g. Musical Instrument Digital Interface (MIDI) or Open Sound Control (OSC), in which case the Score part would just consist of a command to make Csound “wait” for a certain amount of time. Csound operates at two different rates, namely audio rate and control rate. Audio rate is the sampling rate of audio signals. Most common audio rates are 44.1 kHz, which is the sampling rate used for audio CDs, and 48 kHz, which is the sampling rate used for audio tracks on most DVDs. The control rate is the rate at which any sound altering parameter such as amplitude, frequency, filter cut-off etc. is varying. Both rates can be defined arbitrarily, but may be limited by hardware restrictions of the host computer. In addition to an absolute rate, the control rate can also be set to a ratio of the audio rate, which can be useful for some audio signal manipulations. In the SIREN vehicle sound models, the audio rate is set to 44100 Hz and the control rate ratio to 512 Hz, resulting in a control rate of 44100/512=86.13 Hz.

2.4.2 Sound model

There are currently two main sound models implemented within SIREN, corresponding to the two vehicle models implemented in Sim IV where most of the critical testing of SIREN has been performed. The two models are created in a very similar manner, and hence they will be described in general terms valid for both cases.

As basis both for the car sound model and for the truck sound model interior sound recordings were performed during a number of driving cases at the Volvo test track. The driving cases were chosen to cover as many different properties of the sound as possible, such as full throttle acceleration and engine braking at different gears, coasting to stop with engine switched off and idling engine at stand still. Sound was recorded by microphones placed by the drivers’ ears, in accordance with the relevant international standard (ISO, 1980) as well as with a Core Audio Tetramic sound field microphone placed at the passenger seat. Synchronized to the sound signals, Controller Area Network bus (CAN-bus) signals were recorded, providing data on engine speed of revolution, engine torque, vehicle speed and current gear, among other things.

\(^1\) [http://one.laptop.org](http://one.laptop.org)
The recordings were analysed by dividing the signal into short time windows and applying a common Fast Fourier Transform (FFT) to each window. From the resulting time-frequency matrix the most prominent tonal components were identified, and their relation to the engine speed of revolution was determined. The relation to the revolution speed is referred to as engine orders, as in multiple orders of the engine revolutions per minute (rpm) value. An example of this analysis for a truck can be seen Figure 8 with a few engine orders highlighted.

![Figure 8: Engine order analysis using FFT of sound recorded in a Volvo FH12 truck equipped with a Volvo iShift automatic gear box, performing a wide open throttle (WOT) acceleration from stand still to 90 km/h. Engine orders 0.5, 1, 1.5, 2, 2.5 and 3 can be seen marked in blue](image)

The truck has a six cylinder, four stroke engine which means that the firing of the six cylinders is evenly distributed over two revolutions, i.e. three firings per revolution. If the engine is turning at 840 rpm, which is 840/60 =14 revolutions per second (rps) that means there will be a cylinder firing every 1/(14*6/2) = 0.024 s or at a rate of 42 Hz. Since that is three times the rps it is called the third order. A five cylinder, four stroke engine would have a corresponding 2.5th order since it has five firings distributed over two revolutions. These firings will constitute the fundamental frequency of the engine sound, and according to the mathematical definitions behind the Fourier Transform the exact engine sound can be recreated by adding an infinite number of harmonics to that fundamental.

In reality, however, only the most prominent harmonics are important for the perceived sound. Since there are nonlinearities in the system and since there is a gear box introducing odd ratios of the fundamental frequency not only integer multiples of the fundamental frequency are of interest, as would be the case purely from the Fourier definitions, but any prominent tonal component that varies in correlation with the engine rpm. When the harmonics of interest have been identified, the amplitude variation of each harmonic related to measured engine rpm and engine torque is analysed using a two dimensional least squares linear regression. The result is a two
dimensional linear function for each harmonic depending on rpm and torque. While it would be possible to directly apply amplitude values for each harmonic directly from measurements, e.g. through lookup tables, the measurements contain noise from other mechanisms than the firings of the engine cylinders and the amplitudes are hence not directly applicable and some sort of averaging is needed. A least squares linear regression makes for a useful averaging and can also be extrapolated outside of the measured range if needed, thus being more generic than a lookup table solution. The computational cost of evaluating the linear regression model is in essence negligible on a modern computer and there is hence no reason to use a pre-allocated lookup table.

Csound uses functions called Opcodes to manipulate signals. One such opcode is called adsynt2 and uses, as the name suggests, additive synthesis to create desired sound characteristics. Additive synthesis is precisely the inverse of Fourier Transform, i.e. by adding a number of harmonics to a fundamental almost any sound can be created. The same principle has been applied to achieve a wide variety of tonal characters in pipe organs long before Jean Baptiste Fourier formulated his theory, so it is a method proven over long time. In this case it is very well suited for engine sound synthesis since the analysis related to the recorded rpm can be used to synthesise sound related to the virtual rpm from the simulator vehicle model. In Csound the call to the adsynt2 opcode looks like:

```csound
aengineout adsynt2 0.05, kfreqfund, giwave, gifrqs, giamps, icnt, giphas
```

This means in plain text "use an overall amplitude value of 0.05 and values of fundamental frequency, wave shape, harmonic frequencies, harmonic amplitudes, number of harmonics and relative phase of harmonics given previously to synthesise a sound signal that is named ‘aengineout’". The input parameters are defined in a set of tables that are recreated at every change of virtual vehicle rpm or torque, in which case every harmonic is given an amplitude calculated from the two dimensional linear regressions done in the analysis of the recorded sound. In Csound the calculations performed for each harmonic look like:

```csound
kindex = 1
kampharmonic pow 10, ((krpm * -0.00668178 + kload * 0.0697344 - 8) / 20) + tablew kampharmonic, kindex, giamps
```

In addition to the engine sound the most important sound source is the tyre/road noise. In the Csound vehicle models used in SIREN this is modelled by applying two filters to white noise. One of the filters, or rather bank of filters, represents the transfer function from the noise generation between the tyre and the road to just outside the drivers’ ear inside the car cabin, both air borne and structure borne. In reality this transfer function is likely not time invariant, but has been simplified as such to keep the model simple. The transfer function has been identified through difference calculations of interior noise from field measurements and exterior noise measurements. The second filter is a time varying low pass filter that corresponds to the variation in generated frequency spectrum due to the roughness of the road surface and the speed of the vehicle. It allows for different road textures by specifying a Mean Profile Depth (MPD) of the surface and thereby changing the initial low pass cut-off frequency and shelf amplitude of the filter. The cut-off frequency and shelf amplitude will then vary with speed according to a function determined from analysis of recordings. In Csound this looks like:

```csound
knoisemp = 0.125 * sqrt(kMPD) * kvel / 25 ; kbeta is the noise generator high frequency cutoff, 0 means nyquist
kbeta = 0
anoisew noise knoisemp, kbeta
```
First, the amplitude of the noise is calculated from the given MPD value and the given velocity. Second, white noise is generated using the corresponding opcode and the calculated amplitude. Third, the static filter bank corresponding to the time invariant transfer function is applied to the white noise. Fourth, the time varying filter corresponding to the vehicle driving over the texture roughness at a varying speed is applied, and finally the tyre/road noise is combined with the engine sound at appropriate amplitudes:

\[
a_{\text{out}} = 1.5 \times (a_{\text{engineout}} + 12 \times a_{\text{noiseout}})
\]

2.4.3 Implementation

The sound generator has of course to communicate with the sound manager to function properly. Csound supports various means of external control, one of which is Open Sound Control (OSC). OSC uses specifically formatted UDP packets to allow communication between supported devices. The formatting is such that it allows packets to contain a network address and port, a name of the receiving function, the format of the transmitted control signal and of course the control values themselves. In Csound the reception of OSC packets is set up by first initiating a listening port, and then receiving values sent to the corresponding port and the corresponding named receiving function:

```
gilisten OSCinit 47120
kk OSClisten gilisten, "/engine", "fff", krpm, kvel, kload
```

This means in plain text: "Start listening for OSC messages at port 47120. Whenever a new packet arrives, if it is supposed to control the function named ‘/engine’, receive three 32 bit floating point numbers and store them in variables krpm, kvel and kload respectively”. In order to transfer the necessary information to Csound from the simulator vehicle models a small addition to the simulator code has been added that uses rpm, load and velocity data from the vehicle model, formats them as OSC packets using the appropriate control function name (“/engine” in this case) and sends them to the appropriate network address. Since the main program loop of the simulator kernel runs at 200 Hz new OSC packets will be sent to the sound generator at a rate of 200Hz. It is required to implement something similar in any simulator environment where SIREN is to be implemented.

The sound generated by the Csound vehicle sound model is also sent to the sound manager using UDP packets, however not formatted as OSC. It is done by calling the opcode `socksend` which will store audio data in a packet, in the above case in the size of 1024 bytes, and send it to the defined receiver, in this case port 12001 on the same computer, as soon as the packet is filled. With the trailing number 1 argument the format of the signal sent will be 16 bit integers, which is directly compatible with the audio buffers used in OpenAL, instead of the default internal format of Csound, which in most cases is 64 bit floating point data (but which in some cases can be 32 bit floating point data if Csound
is compiled without support for double precision). Conversion from this format to that used in the sound manager is if needed performed by the CsoundOwnEngineStream plug-in, which also is the receiver of the stream. The above use of the socksend opcode is valid for the current release of Csound (5.19) and was added by the Csound developers upon request by the SIREN project.
3 Results

3.1 SIREN Functionality

3.1.1 Supported sound formats

As alut is used SIREN theoretically supports every file format that alutLoadMemoryFromFile can read, but a plug-in for the wav file format is the only format included by default in the SIREN source code. An additional file format that is supported, although not through alut, is the Head Acoustics hdf-format. A simple reader for it has been written which assumes 24 bit, 44100 Hz stereo files. The reason for this file support is that the Head Acoustics binaural recording equipment is commonly used for interior vehicle sound recordings by many of the ViP partners.

SIREN also supports streamed sound. By default it is assumed that the streamed sound is a mono 16 bit sound, but a stereo sound could also be received. When receiving sound from a stream every piece of data is assumed to be sound and thus users of SIREN have to take care that unwanted data, such as a file header, is not sent to the receiving port. A custom version of the stream receiver has been written specifically for receiving streams from Csound and hence converts from 32 bit float data to 16 bit signed integer data.

3.1.2 Spatialization of sounds

The positioning and movement of sound sources are supported natively by OpenAL and this functionality is utilized for spatialization. By giving a sound object a velocity value different from zero OpenAL will automatically apply a Doppler shift to the pitch of the sound corresponding to the relative velocity between the source and receiver. It should be remembered that the position and the speed of a sound object are independent of each other in OpenAL, and thus it is possible for a sound object to have a velocity value causing Doppler shift, while remaining at the same apparent geographical position. In SIREN the velocity and the apparent position of a sound source have been coupled through a simple equation of motion, but can be de-coupled should the need arise. Since stereo sound, like all other multi-channel formats, is a form of spatialization OpenAL will pass-by its internal positioning if receiving stereo sound.

3.1.3 Available plug-ins

CirclingSoundSource: The CirclingSoundSource plug-in can be used as a first test to see that SIREN is working properly and that surround sound is present. This plug-in presents a sound circling around the listener and includes both updating of positions and of velocities.

CsoundOwnEngineStream: As data is sent from the simulation kernel to the sound model in Csound a sound stream is created. This sound stream is sent back to SIREN as UDP packets, and this plug-in receives the stream, converts the data and plays it through OpenAL.

OscEffectSound: A small implementation of receiving OSC sound messages. Messages will be assumed to include the name of the sound and a position for it. The address pattern can be chosen as desired. Upon receiving an OSC message the sound file related to the specified name will be played from the specified position.
StreamToFile: SIREN interprets everything within an incoming stream as a sound and tries to play it as such. This plug-in stores the received sound stream to a file, enabling offline analysis which can be useful for identifying non-audio data in the stream.

PassingBySoundSources: Sound sources are created in front of the listener and are given a speed towards the listener. The sounds then come “driving by” the listener. This plug-in is, as the CirclingSoundSource plug-in, mainly for testing purposes.

VtiNetworkReader: This plug-in implements extracting a sub-set of data when receiving “actor information” from the simulation kernel at VTI, i.e. positions of surrounding vehicles. This plug-in thus presents the surrounding traffic to the driver by positioning sound from surrounding traffic at positions corresponding to those in the visual rendering.

3.1.4 User interface

A user interface, which can be enabled in the configuration file, is present. In this GUI functionality for adding, removing and playing sounds is present as well as possibility to position sounds. The user interface will pre-load sounds specified in the configuration file. This interface is mainly aimed at developing sound scenarios where spatial properties of e.g. warning sounds are of importance.

3.1.5 Implemented sound sources

In general, adding functionality to the simulator software at VTI is done on a per project basis. If a project has a need for some specific functionality it will be added within the time and budget frames of that project. The above mentioned sound model for the engine sound as well as sound files representing sound from surrounding vehicles (currently limited to one sound for cars and one for trucks) have been developed within the project presented in this report. The road noise model, including also models for wet road surface, and cracks and holes in the road, was originally developed for the LjudprogramPersonbil in an earlier project (Bolling, et al., 2013), but has been adapted for SIREN in this project. Patches with a different MPD value and with a number of cracks or holes per unit distance, Poisson distributed, are defined as a property of the road and control the noise models. These have also been adapted for SIREN, but are yet to be experimentally validated in this new implementation.

Some of the sound environment is inherent from the cabin, e.g. noise from the air conditioning equipment, while sounds from diverse electronic systems such as radio or GPS have to be developed as needed. Forward Collision Warning and Lane Departure Warning sounds are included as sound files provided by Volvo and are hence the actual sounds used in the corresponding driver support systems. Table 2 summarizes the functionality currently implemented in the Sim IV installation of SIREN.
Table 2: Sound functionality currently implemented in Sim IV through SIREN

<table>
<thead>
<tr>
<th>Sound</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Own engine</td>
<td>Stream</td>
</tr>
<tr>
<td>Engine crank</td>
<td>File</td>
</tr>
<tr>
<td>Own tyres</td>
<td>Stream</td>
</tr>
<tr>
<td>Potholes / damaged road</td>
<td>Stream</td>
</tr>
<tr>
<td>Wet road</td>
<td>Stream</td>
</tr>
<tr>
<td>Surrounding traffic (engine + tyre/road noise)</td>
<td>Looped file</td>
</tr>
<tr>
<td>Blinker</td>
<td>File</td>
</tr>
<tr>
<td>Principal Other Vehicle Car Horn</td>
<td>File</td>
</tr>
<tr>
<td>Lane Departure Warning</td>
<td>File</td>
</tr>
<tr>
<td>Forward Collision Warning</td>
<td>File</td>
</tr>
</tbody>
</table>

3.2 Computational performance

3.2.1 Using the FiTPC2 with external sound card

During early development there were some problems to configure the operating system on the FitPC2 computer since there were specific device drivers which conflicted with operating system updates. The computer was also seen to be too slow to handle the system at an early stage. With the setup described in section 2.2.2, running the current release of the SIREN sound manager, playing one sound file together with the Csound vehicle sound model, makes the sound manager and Csound use about 15% CPU load each.

3.2.2 Running SIREN in Sim IV

During the preparation for the Sim IV inauguration a performance test was made. In this setup the surrounding traffic and an engine start sound were used. There were also two streams generating engine sound at the same time. One of the streams was sent to the sound manager to be played through the surround system, and the other stream was sent to a script running to check that the data was sent at a correct speed. This was due to a synchronization problem causing Csound to send extra data packets if it detects problems with the real-time performance. The generation of the two streams used approximately 25% of the processing power while the sound management used approximately 30%. Memory usage was 1 to 2%. In this case both the sound management and the sound generation were run on the same computer which is not necessary. However, to ensure uninterrupted and well synchronized streaming data it is recommended to do so, if possible.

3.2.3 Running SIREN at Volvo Trucks

SIREN has also been tested at Volvo Trucks on an Intel P4 2.8 GHz Dual Core computer running a 64-bit Ubuntu 11.04 GNU/Linux operating system. During the test there were 30 to 40 sound sources moving around in the environment. This resulted in approximately 20% processor load. Additionally, Volvo Trucks has also successfully
added a custom plug-in to the SIREN software in order to adopt the management of moving sound sources in the simulated environment to their existing simulator software.

3.3 Audio performance

For validation purposes, measurements inside each cabin using a similar measurement setup as during the field measurement has been performed, proving the feasibility of the method using generated CAN signals from the VTI simulator vehicle model for sound and vibration measurement purposes.

As can be seen in Figure 9, the generated sound levels correspond quite well to the real levels recorded inside a truck, except for the very lowest frequencies where the generated levels are too low. The generated sound was validated by a vehicle dynamics expert at Volvo Trucks with many years of driving experience. A reason for the low frequency discrepancy might be that the vibration levels of the truck cabin in the simulator were not calibrated before the test, which might have caused an over-emphasis on the sound. This discrepancy can easily be remedied by increasing the sound level of the subwoofer in the simulator truck cabin.

![Figure 9: Comparison between a sound measurement in a real truck and a measurement of corresponding sound generated in the Sim IV truck cabin](image)

Also for the car cabin the generated sound levels correspond well with interior sound levels recorded in a real car (cf. Figure 10). There is a slight increase in levels at the mid-low frequency range, which is most likely due to very sensitive microphone placement. The car cabin in Sim IV is in fact half a car cabin with a back wall welded just behind the front seats. The reduced size and the fact that the subwoofer is mounted behind the passenger seat in that back wall mean that placing the microphone just
slightly too close to the back wall or to the right hand side of the cabin cause an increase in measured low frequency levels. Also, the smaller cabin size cause standing waves, so-called room modes, to occur at other frequencies than for a real car cabin. Some further sound equalizing of the speaker system might reduce the difference.

Figure 10: Comparison between a sound measurement in a real car and a measurement of corresponding sound generated in the Sim IV car cabin

3.4 User feedback

Feedback from users has not been collected in a formal way using questionnaires, but rather through informal discussions with people using the system as implemented in Sim IV during demonstrations, scenario development etc. The discussions have included people with simulator experience as well as people working with modelling sound in a simulation environment or working with interior car/truck sound quality. In this setup they were presented sound from surrounding traffic and real-time engine sound generation as new features introduced by the use of SIREN. Other sounds which were also present were an LDW warning sound and an engine start sound. Usually the demonstrated road was a highway with a medium to high traffic density.

The general feedback has mainly been positive, and most people who tried the setup in Sim IV considered the presented sound to be good. Most positive feedback was given to the surrounding traffic which has a big effect on the perceived environment from the driver’s point of view. The generated engine sound was not considered such an improvement over static sound with interpolation, but was considered to sound good.
4 Discussion

4.1 Experience from using SIREN

The positioning of sounds seems to have added a significant layer of perceived realism. This can be seen both from initial comments from drivers as well as from the perspective of a person developing an experiment scenario. After experiencing the sound environment in Sim IV the lack of sound from the surrounding traffic in the sound system seem to be very noticeable when driving in Sim II (and also previously in Sim III). The importance of the spatialization of sound in a driving context has to some extent been addressed in a project involving a precursor to SIREN (Fagerlönn, Andersson, & Liljedahl, 2011) and further research focusing on such issues has been facilitated through the development of SIREN.

The perceived improvement from the real-time generated sound at the inauguration of Sim IV would likely have been bigger if the demonstrated road would have presented a varied topography enabling a more extensive use of torque feedback from the engine.

In general, some initial problems have been solved, some remain, but mainly the SIREN system for sound generation now runs smoothly in Sim IV as well as in Sim III.

4.2 Experience from developing

4.2.1 Common version handling of source code

At the start of the project source code was sent between partners by mail or ftp. As the source code grew it was quickly apparent that problems like “Which files are the most current?” and “We have made changes in the same file, how do we merge?” appeared. To handle these problems the Subversion (SVN) handling system at a server at VTI was used. A problem is that there was no connection to the SVN server from outside VTI and thus a developer from HiQ needed to come to VTI to be able to get the latest updated files and commit changes. This was not an optimal solution as it would have been better to have access to the server from outside VTI, but it was far better than sending files by mail or ftp. So, for every development project that handles source code between partners it is recommended to use a server which every partner involved can access.

As a version handling system SVN worked well. Graphical client systems were tested; TortoiseSVN was used on Windows and RapidSVN was used on Linux. Both these systems worked well for this project. It should be mentioned that during the development of SIREN the source code has been relatively simple and thus the need for advanced functionality from the SVN clients has not been large.

4.2.2 Cross platform development

During the development on different platforms the policy has been that when committing source code to the Subversion (SVN) code-base version handling server the source code has to compile without errors on all target systems. While this policy has largely been followed there were occasions where code compiled under Linux while not under Windows and vice versa. For an individual developer this would result in compilation errors that someone else has introduced, which hence would cause a reduction in development efficiency. Even though handling such situations meant some extra work, the consensus was to resolve any platform dependent issues immediately.
That way the resulting source code was continuously kept portable among the different operating systems used. The pointed out problems are most likely unavoidable when developing on several different platforms and are something that has to be taken into account when planning future projects.

When developing, the work distribution was such that developers had responsibility for specific areas of the source code. This was to minimize collisions of changes in the same source code files between developers. This was seen valuable as the source code was small in the beginning and the possibility of collisions high even with only two developers.

During weekly meetings it was also discussed what had been done and what should be done as a way to keep track of the development. This might not have been that valuable since both developers were mostly at VTI and these meetings could probably have been reduced to once every two weeks. If the developers would have spent more time at different places this might have been more important. These meetings have unfortunately not been documented other than the informal post-it notes used as informal reminders. In retrospect it would have been good to have a road map over the development.

4.2.3 Code documentation

The documentation proved to have both positive and negative effects on the developing. On the positive side the tool doxygen appeared to be able to generate the documentation. On the negative side there was a larger workload when functions were changed and the documentation needed to be updated. There is also the problem of how to handle documentation of external libraries or other external source code. If such a source is documented and later needs to be updated the documentation is usually wrong, thus it will take time to adapt or remake the documentation. The same problems apply to code that changes a lot since a lot of extra work is required if the code always should have updated documentation. Another aspect of documenting the source code is that a lot of extra text is introduced in the source code. Simple functions that are self-explanatory do not benefit from extra documentation cluttering the source code. The conclusion is that there is no real reason for documenting source code that changes in a standardized way. A short comment in the source code would be sufficient and a “wiki” type documentation could be used as a general description of the software.

During development it was seen that running Csound separately from the sound management was useful since it results in the sound generation and sound management being two separate programs running independently. While practical when developing, this should not be the case in the future. The idea is that SIREN keeps track of them both and starts sound generation when needed and thus the sound management may start up Csound threads, or any other thread generating sound. This procedure has been tested but not yet been implemented in a general way, which is a development issue that should be taken care of in the future.
4.2.4 Use of external libraries

The experience of external libraries is that the amount should be minimized. Every library needs to be integrated on every platform where it is to be used and such integration takes time. Also, every time a new external library needs to be updated it needs to be tested again on every platform. This is time that could be spent on development instead.

External libraries also take up space. Thus, if a lot of the library content is not used the library is using unnecessary resources and adds to the time required for compiling. For instance Qt, which was used as development environment, has worked well but the library might be unnecessarily large compared to the needs. This is mainly due to the fact that the use of a graphical user interface to SIREN has been diminishing, and if it is not to be used at all, Qt will mainly be used for network handling and the “makefile” setup. In such a situation it might be better to use another, smaller, cross platform “makefile” system and another network handling.

Other libraries which currently are not used much and probably should be removed are Alut and Boost.

4.2.5 Open Source and special needs

During the development of SIREN a lot of effort has been put into finding suitable software on which to base sound generation, spatialization etc. Since very little existing software is intended to be used in the way employed by SIREN, some functionality was relatively scarcely documented, or not even implemented at all. Csound was chosen mainly because of the following reasons; it is free to use under GNU/GPL, it is controlled by commands collected in a plain text file, it has built in support for Open Sound Control, and it has built in support for audio streaming over standard UDP network protocol. In early testing, identifying the format of data in the stream sent by Csound was difficult as it was originally intended to be received by another instance of Csound, and hence the format would be the same in both ends. Also, there were intermittent, but persistent, synchronization problems when controlling Csound using OSC and receiving output using UDP data streams. After extensive trial and error it was found that Csound needs to synchronize to an external clock, e.g. a sound card, to achieve stable performance. Since no output was being sent to any sound card synchronization was not achieved. This was finally solved by defining audio inputs in Csound so that synchronization was achieved without the need to output sound to a sound card. Similar problems with hardware dependence on different development systems were discovered and solved only after extensive browsing through the scarce documentation. On the other hand, a lot of support has been given from the Csound development community, resulting in added functionality to the Csound source code allowing for formatting of sound stream data to fit the format later used by the OpenAL audio data buffers.

4.3 Future development of SIREN

The modular design of SIREN allows for a relatively easy extension of features, but there are still some of the basic parts of SIREN that could benefit from development. Sending sound streams from Csound to the SIREN sound manager as UDP packets works, but support in SIREN for the cross platform, real-time, low latency audio routing daemon Jack Audio Connection Kit (JACK) is recommended as a more general
application. Using JACK instead of the built-in UDP sender in Csound would solve the problems experienced with synchronization between Csound and OpenAL, as everything would then be synchronized against JACK which in turn synchronizes against the hardware. This would also facilitate using other sound generators if needed since JACK is widely supported by many Open Source audio applications.
5 Conclusions – What has been provided through SIREN?

A number of representatives from different ViP partners were involved in the project as a reference group. Feedback from this reference group states that a number of sound properties are of interest, such as influence of power train sound on perceived comfort, monotony in sound as a factor in driver drowsiness, spatial properties of warnings to enhance their efficiency etc. Most of these issues are restricted by limits in weight and cost of implemented systems in real vehicles, and hence there is a large need for prioritization. SIREN promises a reliable way of performing this prioritization by allowing the partners of ViP to test properties of different solutions within a simulated environment.

In its current state the apparent difference between SIREN and previous sound generating software is mainly the presence of sound from surrounding traffic. The large benefit however is the way SIREN allows for continuous improvement of the sound environment. For example, in order to add sound of a completely new vehicle all that is needed are recordings of a number of driving cases with included measurements of vehicle speed, engine torque and speed of revolution. From that a model of the sound can be created and implemented into the existing software in a manner almost identical to the existing models, hence minimizing the effort. Also, the properties of existing vehicle models may be easily updated. Through the use of real time synthesis the sound can be updated on the fly, allowing for quick evaluation when developing new or altered models. Additionally, SIREN allows for continuous variation of simulation scenario parameters such as road surface texture to be accounted for in the sound presented to the driver, thereby increasing realism through e.g. perception of velocity.

The relatively easy sound model expansion has allowed for recent additions used in demonstrations and experiments, e.g. a very realistic road noise model used in a driver drowsiness experiment and the sound in an electric car presented in a demonstration. Plans for future model development include a model for a hybrid diesel-electric heavy truck for a project about electric highways, and a model of noise reduction after grinding a worn asphalt road surface.
References


ViP is a joint initiative for development and application of driving simulator methodology with a focus on the interaction between humans and technology (driver and vehicle and/or traffic environment). ViP aims at unifying the extended but distributed Swedish competence in the field of transport related real-time simulation by building and using a common simulator platform for extended co-operation, competence development and knowledge transfer. Thereby strengthen Swedish competitiveness and support prospective and efficient (costs, lead times) innovation and product development by enabling to explore and assess future vehicle and infrastructure solutions already today.