

VAEBuilder - A General Purpose Software Tool for the Description and Implementation of Virtual Audio Environments

January 19, 2000

Mark Pearson

Abstract

This document describes the issues relating to the design and implementation of a software tool for creating and exploring arbitrary virtual audio environments (referred to hereinafter as VAEs). A prototype package called VAEBuilder has been developed with some of the functionality described in this document. VAEBuilder is implemented in Visual C++ using MFC and comes with full hypertext or printed documentation for the sources. It is a work in progress, as many of the features and ideas described in this document have yet to be fully worked through or implemented, but even in its present state it acts as an example of how the various hardware and software technologies might be integrated.

Table of Contents

Abstract	i
1 Introduction	1
2 Feature Wish List	2
3 User Interface Wish List	3
3.1 Scripting Languages	3
3.2 A GUI for visualising and manipulating VAEs	3
4 Application architecture	5
4.1 Document/View Architecture	5
4.1.1 The View	5
4.1.2 The Document	6
4.2 Overview of classes provided	7
4.2.1 OpenGLView	7
4.2.2 VAEBuilderView	7
4.2.3 VAEBuilderDoc	7
4.2.4 HuronTCPWrapper	7
4.2.5 VAEHuronSocket	8
4.2.6 Sound	8
4.2.7 Room	8
4.2.8 Door	8
4.2.9 Voice	9
4.2.10 SpaceBall	9
5 Overview of Huron Terminology	10
5.1.1 Workspace	10
5.1.2 Wave	10
5.1.3 Voices	10
5.1.4 Communication Channels	11
5.1.5 Acoustic Spaces	11
5.1.6 Rooms	11
5.1.7 Room Segments and Doors	11
5.1.8 MIDI Messages	11
6 VRML97	12
6.1 Definitions used in VRML97	12
6.1.1 node	12
6.1.2 grouping node	12
6.1.3 scene graph	12
6.1.4 sensor node	12

6.1.5	geometric property node	13
6.1.6	geometric sensor node.....	13
6.1.7	geometry node.....	13
6.1.8	interpolator node	13
6.1.9	route	13
6.1.10	event.....	13
6.1.11	event cascade	13
6.1.12	eventIn.....	13
6.1.13	eventOut.....	13
6.1.14	prototype	13
6.1.15	scripting language	13
6.2	VRML Parser.....	14
6.2.1	Relationship between Room file formats used by Lake and VRML nodes	14
6.3	Prototype sensors.....	14
7	Possible VAE demos	15
8	Possible Application Areas	16
8.1	Aural presentation of ordance survey maps.....	16
8.2	Spatial granular synthesis	16
9	Appendix.....	17
9.1	External information resources and programs used in the development of VAEBuilders	17
9.1.1	Doxygen – a tool for automatically documenting C++ code.	17
9.1.2	CVS – a system for version control of source files.....	18
9.1.3	Python – the object-oriented scripting language	18
9.1.4	OpenGL for Windows resources.....	18
9.1.5	Windows Sockets Programming Resources.....	19
9.1.6	Spaceware SDK and the SpaceBall 3-D input device	19
9.2	Coordinate Systems and Matrix Conventions.....	19
9.2.1	SpaceBall coordinate system.....	20
9.2.2	OpenGL coordinate system.....	20
9.2.3	LakeDSP coordinate system	21
9.3	Steps for setting up the Dell Dimension XPS R400	21
9.3.1	Include and library paths in Visual C++	21
9.4	Rebuilding the documentation for VAEBuilders	22
9.5	Brief instructions on using VAEBuilders.....	22

1 Introduction

This document describes the issues relating to the design and implementation of a software tool for creating and exploring arbitrary virtual audio environments (referred to hereinafter as VAEs). A prototype package called VAEBuilder has been developed with some of the functionality described in this document. VAEBuilder is implemented in Visual C++ using MFC and comes with full hypertext or printed documentation for the sources. It is a work in progress, as many of the features and ideas described in this document have yet to be fully worked through or implemented, but even in its present state it acts as an example of how the various hardware and software technologies might be integrated.

On the subject of hardware and software the set-up used for development was as follows:

- 400MHz Pentium II PC running Windows NT.
- Lake Huron CP4 convolution processor for the spatial audio.
- LabTec Spaceball 3003 FLX 3-D input device.
- Visual C++ 6.0 and MFC.

The current version of VAEBuilder allows a simple baton-like object with two sounds, one attached to either end, to be moved about and oriented in 3-D space using the Space Ball input device. As the object is moved around its position is visually rendered using 3-D graphics and sonically rendered using the Huron spatial audio processor. The sounds associated with either end of the 'sound baton' are simply triggered periodically.

2 Feature Wish List

Before describing VAEBUILDER itself this section looks at the features a general purpose 3-D audio environment building package would ideally have. This feature set is driven by the overall goal of being able to describe arbitrarily complex environments which have the potential for user navigation and interaction. To realise this goal the following features were deemed essential:

- Ability to spatialise sounds within virtual acoustic environments using the Lake Huron spatial audio server controlled via a TCP/IP link
- Integration of the SpaceBall 3-D input device
- An architecture open and flexible enough to deal with the following aspects of a VAE:
 - Spatial positioning of sonic objects
 - Behaviours of sonic objects – some sounds might be ambient helping to create an overall atmosphere whilst other might be triggered by specific user actions and others might have behaviours of their own or act as navigational beacons.
 - Support for user interaction with objects in an environment
 - Support for time domain scripting of events
 - The ability to express arbitrary rules relating objects in an environment to each other and to the observer (either mathematical, based on the laws of physics or more abstract). Hierarchical relationships between objects would be represented with a *scene graph* - a tree structure expressing parent-child relationships of the objects in an environment.
 - The ability to describe and model acoustic spaces
- A graphical user interface allowing the environments to be visualised, since working with any 3-D objects is difficult without some kind of visualisation (for the designer), even if the intention is to render the final product purely in 3-D audio.

A number of important issues come out of these requirements including the choice of an appropriate application framework and GUI toolkit, the data structures used to represent a VAE, choice of a suitable scripting language and a suitable 3-D graphics API.

Of this set of requirements the following subset are actually implemented in VAEBUILDER:

- An (incomplete) C++ API allowing the Lake DSP CP4 convolution processor to be controlled via a TCP/IP link.
- Integration of the LabTec SpaceBall 3003 FLX 3-D input device.
- A multiple document interface (MDI) architecture built using Visual C++/MFC.
- Basic OpenGL functionality allowing any VAE elements to be visualised.

The features described in the above list are not completely finished but in conjunction with this document and the VAEBUILDER Class Library Reference the system should prove a useful starting point for further work in this area.

3 User Interface Wish List

The main tools available for describing 3-D worlds include 3-D computer animation packages and the *Virtual Reality Markup Language* (VRML). Many animation packages provide sophisticated GUIs for constructing the 3-D worlds and provide techniques such as key-framing and motion curves for predefining the ways in which the objects evolve over time. But for more sophisticated control over object behaviours a *scripting language* is often used. This allows the time-domain behaviour of objects to be expressed in terms of algorithms rather than simple static curves. The use of a general purpose programming language to describe the behaviour of objects provides maximum flexibility as it can be used to describe just about any kind of relationship that might be useful. Examples of scripting languages include Tcl, Python, and Perl etc.

The user interface should therefore include the following elements:

- A GUI allowing the VAEs to be visualised and directly manipulated in 3-D.
- A scripting language for describing for describing elements of the VAE's behaviour which are not easily expressed using a GUI.

3.1 Scripting Languages

Possible candidates are:

- VRML97 – strictly speaking not a scripting language but it is included here since it provides a specialised, high-level text-based language for expressing interactive 3-D worlds.
- A custom designed language using the parser-generator tools Lex and Yacc.
- A high level OO scripting language such as Python

Regardless of the choice of language the final product would ideally be capable of the following:

- Placing objects statically at given locations
- Scheduling events for occurrence at pre-defined times
- Describing relationships between the user's position, the various object positions and the resulting behaviour of those objects
- Describing trajectories through 3-D space
- Making the VAE respond in an appropriate manner to user interaction/exploration

There are obvious advantages to adopting an existing language such as VRML97 as a description language, not least because many of the technical hurdles relating to describing complex, interactive 3-D environments have already been solved. This would probably involve building or using an existing VRML parser and writing custom C++ code to implement any new features required which are not already a part of VRML. For a more detailed discussion on the subject of VRML see section 6.

3.2 A GUI for visualising and manipulating VAEs

An essential component of the tool would be the ability to visualise any VAE's constructed. This could be extended to direct manipulation of VAE components using a GUI. VAEBuilder uses OpenGL for its 3-D graphics since it is the industry standard API and provides built-in functionality for selecting and manipulating 3-D objects. In order to provide this feature a new *view* class called `OpenGLView` has been implemented. This provides the basic functionality for

creating an OpenGL canvas within an MFC view window. It also provides basic virtual functions for initialising OpenGL (clear colour, projection mode, double buffering etc.) and drawing the scene. These may be overridden by a derived class to implement the particular graphics required. See section 4.2.1 for details about class `OpenGLView` and section 4 as a whole for details about VAEBuilder's architecture.

Apart from the direct manipulation of 3-D objects the other main area for consideration is the choice of a general GUI toolkit using the familiar buttons, dialog windows, menus etc. The most obvious candidates for this on Windows NT are MFC or Qt – a cross platform GUI toolkit which provides much the same features as MFC, but also runs on UNIX systems. In the end MFC was chosen for VAEBuilder since it is integrated with the Visual C++ development environment and is the predominantly used toolkit on Windows operating systems.

4 Application architecture

This section describes the overall architecture of VAEBuilder as it stands at the time of writing. For detailed documentation on the C++ API see the *VAEBuilder Class Reference* document in the `html` folder of the package. The application has been designed in such a way that it could become a generalised tool for constructing and experimenting with arbitrary virtual audio environments. It should be noted that whilst parts of this document describe features which are actually implemented in the current version, much of the document is concerned with sketching out the intended structure of the finished product. Parts of the system which are still at the ideas stage and have yet to be designed and implemented include:

- A data structure for representing objects, behaviours, scene graphs etc. in a VAE
- Integration of a scripting language (VRML, Python etc.)
- A GUI for direct 3-D manipulation and editing of VAEs

However the following functionality has been implemented:

- Basic application architecture using document/view approach (see section 4.1)
- 3-D graphics visualisations
- TCP-based control of the Lake Huron CP4 spatial audio processor
- Integration of the LabTec Spaceball 3003 FLX 3-D input device

4.1 Document/View Architecture

VAEBuilder is implemented in Visual C++ using the Microsoft Foundation Classes (MFC) and is based around the *document/view* architecture.

A detailed description of MFC application architecture is outside the scope of this document but, very briefly, the philosophy behind the design of a document/view application is that all the data for a particular entity, whether a word processor document, a 3-D scene in an animation package, or a spreadsheet etc. is stored in a class called the *document* class. The document is presented to the user with the aid of a *view* class and all interaction between the user and the document is mediated by this class. Finally there are two fundamental application types, SDI (single document interface) and MDI (multiple document interface). SDI applications only allow one document to be open per instance of the application. MDI applications on the other hand allow several documents to be open simultaneously. A good example of an MDI application is MS Word97.

VAEBuilder is an MDI application and as such is structured such that several VAEs could be open at any one time and the user would be able to switch between them and activate the selected environment for testing. MFC also provides the functionality required to load and save documents by serialising all the data members of the document object to disk and being able to reconstruct the objects and their internal states at a later date. However this feature has yet to be implemented in VAEBuilder since the design of an appropriate data structure needs to be finalised before the code to save and load the data structure can be written.

4.1.1 The View

The view is responsible for catching user input via the keyboard, mouse or other input devices such as the Spaceball, calling the appropriate document member functions to update the state of the VAE, and providing the visual representation of the VAE. It must therefore:

- Be OpenGL capable for rendering the environment using 3-D graphics.
- Provide member functions for handling Windows messages generated by input devices.

VAEBuilder's main class is called `CVAEBuilderView` and is derived from the specially designed generic `OpenGLView` base class.

4.1.2 The Document

The document acts as a central repository for all the data which defines the state of a VAE at any moment in time. Ideally this information would include such items as:

- Objects
 - Positions (room number and coordinates)
 - Velocities/accelerations etc.
 - Other state variables particular to each object
 - Relationships to other objects and/or listener
 - Object hierarchies (scene graph)
- The Listener
 - Position (room number and coordinates)
 - Velocity/acceleration etc.
 - Other state variables
- Data files needed by the environment
 - Huron workspace file (*.wsp) (see section 5.1.1)
 - Sounds
 - Locations of sound files (see section 5.1.2)
 - Associated WaveID's (see section 5.1.2)
 - Associated voice numbers (see section 5.1.3)
 - Acoustic spaces
 - Locations of data files (*.geo, *.sim) describing the acoustic space (see section 5.1.5)
 - Associated room numbers (see section 5.1.6)
 - Door locations and positions, and pointers to adjoining rooms (see section 5.1.7)

One of the items mentioned in the list above refers to relationships between objects in the environment and/or the listener. To clarify what this means some examples of typical relationships are given below:

- Linking objects into parent/child hierarchies
- Expressing the positions, velocities and other more sonic attributes of objects in terms of physical relationships so that the objects move and behave in a complex but natural manner.
- Triggering object events based on the user's proximity to an object.
- Predefining motion paths for objects.

The Virtual Reality Markup Language VRML already contains a number of constructs for expressing such relationships (see section 6). Precisely how much support VRML could provide for arbitrarily complex relationships is a matter for further investigation. But at the very least it would be useful if the internal representation of an environment used by the document could be saved in VRML format.

4.2 Overview of classes provided

This overview is designed to explain in plain English the classes provided by VAEBuilder. For more detailed information see the *VAEBuilder Class Reference* in VAEBuilder's [html](#) folder. The HTML documentation describes the classes and their members functions and variables in detail and also contains cross references to the Microsoft Developer Network documentation provided on CD with Visual C++ whenever MFC or Win32 API concepts are touched upon.

4.2.1 OpenGLView

This class is a general purpose OpenGL view class. It encapsulates and hides from the application the details of setting up an OpenGL based view in Windows NT. It includes functions for setting the video mode (INDEXED/RGB, 256 colours, 16 million colours, single/double buffer mode) and virtual functions for initialising the OpenGL state and drawing the scene. *OpenGLView* is designed to be overridden by a class providing application-specific graphics capabilities. By itself it doesn't provide any functions for drawing a scene but merely sets everything up so that a derived class can concentrate on just that.

NOTE: At the moment this class is actually hard-wired to request 24 bit colour (full RGB) and enable the depth buffer so that hidden face removal works.

4.2.2 VAEBuilderView

This is VAEBuilder's *view* class (see MSDN library under 'Using Views'). An MFC application can actually have several different views on a document, but in VAEBuilder there is currently only one, an OpenGL-based 3-D visualisation window. This class is derived from class *OpenGLView* (see section 4.2.1).

4.2.3 VAEBuilderDoc

This class is VAEBuilder's *document* class and should be the place where all state information about a VAE is stored. In this way the whole set of information can potentially be loaded, edited and saved conveniently by deriving all object classes from *CObject* and then overriding the *CObject::Serialize* function for each. For further information about documents and how to load and save them see the MSDN library under 'Using Documents'.

NOTE: As it stands the *VAEBuilderDoc* doesn't actually contain much information. In fact in the current implementation it only ever contains one object of class *SoundBaton*. There is no load or save or edit functionality currently implemented, but nevertheless the framework is in place for such functionality to be added.

4.2.4 HuronTCPWrapper

This class provides an object-oriented interface which encapsulates all the TCP based commands which the Huron audio server (see section 6-97 of the Huron manual) is capable of responding to. Communication between the VAEBuilder client application and the Huron is via Windows sockets at the lowest level but this class deals only with sending the appropriate TCP command packets which actually control the Huron. This is achieved with an instance of the lower level class *VAEHuronSocket*, described below.

4.2.5 VAEHuronSocket

This class provides a socket connection from the VAEBuilder client application to the Huron audio server. It hides the details of how this connection is implemented using Windows sockets and provides a number of functions for:

- Setting the IP address of the Huron server
- Opening and closing a connection
- Detecting whether a connection has been established
- Sending and receiving packets of information

This class only deals with low level socket interactions. For a higher level interface which has more knowledge of the actual information which is likely to be communicated from the client to the server see the `HuronTCPWrapper` class described above.

4.2.6 Sound

This class encapsulates the concept of a sound object. Such an object has a number of attributes as described below:

- Sound file name
- Voice number
- Location channel
- Position (x,y,z)
- Orientation $(xangle,yangle,zangle)$

4.2.7 Room

This class is intended to represent the Huron concept of a room. The implementation of a room involves the following information:

- A description of the acoustic space (see section 6-115 of the Huron manual), which is represented by *.geo and *.sim files on disk.
- *Room number* – a unique numerical identifier.
- Descriptions of any doors connecting this room to another room.

If an appropriate scene hierarchy data structure were in place then each `Room` object would probably maintain a list of objects in the room amongst other information.

NOTE: This class has not been implemented yet.

4.2.8 Door

This class is intended to represent the concept of a door, which in Huron terms is a connection between two room segments. The information stored in an instance of this class includes:

- Room number of the first room
- Room number of the second room
- Coordinates of corners of door on both adjoining walls.

NOTE: This class has not been implemented yet.

4.2.9 Voice

The Huron audio server can theoretically use up to 72 simultaneous voices. This limit is set by a dialog option in the WavePlayer module of the Huron VRack application. The Voice class helps to make allocation and management of the available voices easier by providing functions to do the following:

- Allocate and deallocate voices
- Trigger and release (silence) voices
- Determine whether a particular voice is free or not
- Create an array of Voice objects to server as a pool for allocation to sound objects.

4.2.10 SpaceBall

This class encapsulates the functionality of the SpaceBall 3-D input device. It provides member functions for initialising the SpaceBall, linking it to a particular target window for input messages to be processed, setting the sensitivity for the six degrees of freedom and providing access to the translation and rotation data generated by the user's movements of the device.

This class has a static member of type SpaceBall* which is the sole instance of the class and serves as the gateway for all other objects to communicate with the SpaceBall input device. This object is thus a *singleton class* – a class of which there is only ever one instance but which is widely used by other objects in an application.

NOTE: Some of the SpaceBall functionality has yet to be implemented such as providing functions to be called whenever buttons are pressed on the SpaceBall. At the moment the SpaceBall class implements reading rotational and translational information from the ball itself and storing this information in matrix form ready for use in conjunction with the SpaceWare SDK math library matrix functions.

5 Overview of Huron Terminology

The Lake DSP CP4 convolution processor is controllable via a TCP based interface allowing the construction of client software packages which may drive the Huron server remotely. All the major functions of the Huron software such as starting the virtual audio rack program *VRack*; loading workspaces into the virtual rack; loading *.sim* files into the *WavePlayer* module and assigning them to voices; and positioning and triggering voices are controllable via this interface.

A list of some of the main terminology relating to the workings of the Lake Huron software is given below.

- Workspace
- Wave (sound data)
- Voice
- Room
- Acoustic space
- Room segment
- Door
- Communication channel (location, angle, etc.)
- MIDI message

5.1.1 Workspace

The top level application on the software side of the Huron CP4 is called *VRack*. This represents a virtual audio rack with rack mountable modules. Examples of modules include *Patchbay*, *IO Manager*, *MultiScape* (a self-contained 3-D sound modelling application), and *WavePlayer*. At any one time the *VRack* application can have a number of different modules installed. A particular configuration of modules and module settings can be saved in a *workspace* (*.wsp*) file at any time.

5.1.2 Wave

The term *wave* is used to refer to audio sample data loaded into the Huron audio server. The term is slightly misleading as the format which the Huron server can read is actually **.sim* rather than the more widely accepted **.wav* standard as might be expected. A command line utility called *wavtosim* is provided for converting Microsoft WAV files to Huron SIM files. Once a set of audio sample data has been loaded it is uniquely identified by a *waveID*. The following is a list of TCP command names used to deal with waves.

- Loaded into CP4 using *HuronLoadWave* TCP command
- Removed from CP4 using *HuronRemoveWave* TCP command
- Linked to a *voice* using *HuronUpdateVoice* TCP command
- Information retrieved using *HuronGetWaveInfo* TCP command

5.1.3 Voices

Whilst a *wave* represents the actual audio sample data stored on disk, the CP4 uses the concept of *voices* to internally trigger sounds in a virtual audio environment. A voice must be associated with a particular set of wave data before the sound can be triggered. This is achieved through the *HuronUpdateVoice* command which takes a *voice number* and *waveID* and associates the former with the latter. The *WaveID* is specified when the *HuronLoadWave* command is issued and it becomes associated with the particular sound file being loaded.

5.1.4 Communication Channels

A communication channel is used to send location and orientation messages to individual voices. This is achieved with the use of the `HuronLocation` and `HuronAngle` TCP commands. A voice is associated with a particular location and angle channel by setting values in the `MultiScape VRack` module window.

5.1.5 Acoustic Spaces

An acoustic space is generated using the `MakeRoom` utility and is subsequently represented by a single `*.geo` file and a set of `*.sim` files co-located in the same directory. The `*.geo` file represents the geometry of the room and the four `*.sim` files represent x , y , z and w components of the generated impulse response.

When loading an acoustic space into the `Huron` using the `HuronSetAcousticSpace` command the data is associated with an *acoustic space ID*.

5.1.6 Rooms

A *room* is the CP4's way of referring to a set of geometry and impulse response data. Once the data for an acoustic space has been loaded and associated with an *acoustic space ID*, this ID is associated with a *room number* using the `HuronSetRoom` command.

5.1.7 Room Segments and Doors

The CP4 is capable of loading a number of different acoustic spaces and then treating them as a series of interconnected *room segments*. This achieved in conjunction with *doors*. A door connects two room segments and may be positioned anywhere between the two extremes of fully open and fully closed. When a door is gradually opened, sounds from the adjoining room are allowed to leak into the current room, modelled as a point source originating from the direction of the door relative to the observer.

5.1.8 MIDI Messages

The main MIDI messages to deal with are `NoteOn`, `NoteOff`, `Program Change` and `Controller` messages.

6 VRML97

VRML (Virtual Reality Markup Language) is a text based language for describing virtual environments. It provides a number of *nodes* for representing different objects in a scene. A scene is represented overall by a hierarchical scene graph. In a VRML file this hierarchy is represented as nested nodes. Before going any further with a discussion about the suitability of VRML97 as a description language for VAE's it is probably a sensible idea to define some terminology used. The next section does exactly this.

6.1 Definitions used in VRML97

The following definitions are taken from the official VRML97 specification document located at <http://www.vrml.org/Specifications/VRML97>. This document is copyrighted as follows:

Copyright © 1997 The VRML Consortium Incorporated. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole, without restriction of any kind, provided that the above copyright notice, this paragraph and the title, URL, and authors of the document as referenced below are included on all such copies and derivative works. However, this document itself may not be partially copied, excerpted or modified in any way, such as by removing the copyright notice or references to the VRML Consortium, except with written permission from the VRML Consortium or as needed for the purpose of developing VRML standards in which case the procedures for copyrights defined in the VRML Consortium standards process must be followed.

TITLE: ISO/IEC 14772-1:1997, Virtual Reality Modeling Language, (VRML97)
URL: <http://www.vrml.org/Specifications/VRML97>
AUTHORS: Rikk Carey, Gavin Bell, Chris Marrin

6.1.1 node

The fundamental component of a scene graph in ISO/IEC 14772. Nodes are abstractions of various real-world objects and concepts. Examples include spheres, lights, and material descriptions. Nodes contain fields and events. Messages may be sent between nodes along routes.

6.1.2 grouping node

One of a set of node types which include a list of nodes, referred to as its children nodes. These children nodes are collected together to share specific properties dependent on the type of the grouping node. Each grouping node defines a coordinate space for its children relative to its own coordinate space. The children may themselves be instances of grouping nodes, thus forming a transformation hierarchy.

6.1.3 scene graph

An ordered collection of grouping nodes and other nodes. Grouping nodes, (such as LOD, Switch, and Transform nodes) may have children nodes.

6.1.4 sensor node

A node that enables the user to interact with the world in the scene graph hierarchy. Sensor nodes respond to user interaction with geometric objects in the world, the movement of the user through the world, or the passage of time.

6.1.5 geometric property node

A node defining the properties of a specific geometry node. A node of type Color, Coordinate, Normal, or TextureCoordinate.

6.1.6 geometric sensor node

A node that generates events based on user actions, such as a mouse click or navigating close to a particular object. A node of type CylinderSensor, PlaneSensor, ProximitySensor, SphereSensor, TouchSensor, VisibilitySensor, or Collision.

6.1.7 geometry node

A node containing mathematical descriptions of three-dimensional (3D) points, lines, surfaces, text strings and solids. A node of type Box, Cone, Cylinder, ElevationGrid, Extrusion, IndexedFaceSet, IndexedLineSet, PointSet, Sphere, or Text.

6.1.8 interpolator node

A node that defines a piece-wise linear interpolation. A node of type ColorInterpolator, CoordinateInterpolator, NormalInterpolator, OrientationInterpolator, PositionInterpolator, or ScalarInterpolator.

6.1.9 route

The connection between a node generating an event and a node receiving the event.

6.1.10 event

A message sent from one node to another as defined by a route. Events signal external stimuli, changes to field values, and interactions between nodes. An event consists of a timestamp and a field value.

6.1.11 event cascade

A sequence of events initiated by a script or sensor event and propagated from node to node along one or more routes. All events in an event cascade are considered to have occurred simultaneously.

6.1.12 eventIn

A logical receptor attached to a node which receives events.

6.1.13 eventOut

A logical output terminal attached to a node from which events are sent. The eventOut also stores the event most recently sent.

6.1.14 prototype

The definition of a new node type in terms of the nodes defined in this part of ISO/IEC 14772.

6.1.15 scripting language

A system of syntactical and semantic constructs used to define and automate procedures and processes on a computer. Typically, scripting languages are interpreted and executed sequentially on a statement-by-statement basis whereas programming languages are generally compiled prior to execution.

6.2 VRML Parser

Various VRML parsers are available in source form, some free others licensable. A VRML parser could be used as the starting point for building a slightly modified VRML to deal with the specific issues relating to VAE's. Parsers found to date include:

- ftp://ftp.sics.se/pub/dive/vrml_parser.tar.gz

This parser uses the compiler tools *Lex* and *Yacc*. This could be incorporated into VAEBuilder so that VRML97 files could be parsed. Some kind of internal representation would have to be developed, relating to VAEBuilder's other object types.

6.2.1 Relationship between Room file formats used by Lake and VRML nodes

The output files produced by the Lake program *MakeRoom* (**.geo*, **.sim*) contain text based information about the dimensions of the room. This information could be translated from appropriate node information encoded in VRML97. In this way the acoustic spaces, sound objects, and behaviour of the whole environment could be coded withing a single VRML file. Another possibility is the use of PROTO nodes in VRML. These allow experimental features to be added to VRML in order to enhance its capabilities. It may be possible to use a custom designed set of PROTO nodes to represent functionality required for VAE's which is not catered for in the existing definition of VRML97.

6.3 Prototype sensors

The existing sensors provided by VRML97 include:

- Cylinder sensor
- Plane sensor
- Proximity sensor
- Sphere sensor
- Time sensor
- Touch sensor
- Visibility sensor

It may be fruitful to take these sensors as a starting point to see if there are similar prototype sensors which could be developed specifically for use in VAE's. For example, the visibility sensor could be replaced with an audibility sensor.

7 Possible VAE demos

- Navigating around a VAE consisting of several objects arranged on a horizontal plane at random but fixed positions, each acting as a 'beacon', emitting a fixed sound at regular intervals. The observer is able to navigate around the environment using the mouse as an accelerator/steering device. This example would allow a lot to be learnt about navigation strategies in 3-D spaces.
- One sound source whose position is determined by a physical simulation of the sound object being attached to a piece of elastic. The other end of the piece of elastic would be attached to the mouse in some way allowing the user to interactively 'swing' the sound round in real-time. This example would allow exploration of the realism of spatial and physical cues in virtual audio environments.
- An audio-realistic environment including other spatial cues such as filtering effects, reverberation, early reflections, doppler shift.
- Field effect microphone recordings of real environments enhanced with added synthesised information.

8 Possible Application Areas

8.1 Aural presentation of ordnance survey maps.

This might be a fruitful area for investigation for several reasons:

1. Maps are spatially distributed diagrams containing symbolic or iconic representations of various objects.
2. The key provided by a map limits the possible elements which may occur in any one map to a finite numbers of possibilities. This narrows the scope of the problem of finding appropriate sonic representations for any given map (without simplifying the problem too much).
3. Any given sonification scheme could be tested on many different maps all adhering to a single standard of information presentation.
4. There is much scope for trying different sonic representation schemes for each symbol containedf in the map's key, or even encoding information over and above what is available on the map. Multimedia educational systems already exist for sighted users, combining maps with further geographical, political, economic and demographic information.

8.2 Spatial granular synthesis

Distributing grains of sound around the listener and using changes in grain density/characteristics to highlight movements.

One of the limitations of sonic as opposed to visual information is that sound is fundamentally temporal rather than static and inhabits a very different space from that of visual perception. One possible way of getting around this limitation is to replace static visual information with sonic information which is dynamic in one sense but static in another. To explain this contradiction more clearly we will introduce the term *granular synthesis*. Granular synthesis is a technique whereby complex dynamically evolving sounds are built up from elementary grains or particles of sound. Various parameters such as the grain length, grain density and other characteristics may be varied under the control of some high level algorithm, and with appropriate control strategies the technique is capable of emulating natural environmental sounds such as those produced by wind or flowing water. Granular synthesis is able to produce sounds which have very complex temporal and spectral characteristics, and combined with the ability to locate individual grains at arbitrary points in space there would probably be more than enough parameters to play with for the sonification of complex, evolving 3D data sets.

One of the advantages of granular synthesis for the presentation of information more suited to the visual domain is that a dense granular sound, whilst varying temporally at the microscopic level can in fact be perceived as an static stream. Salient features could then be superimposed on this static equilibrium. This relies on the fact that when the density of grains reaches a particular threshold, they begin to be perceived as a single event rather than as discrete sounds. By spatially distributing a granular texture using ambisonic encoding it might be possible to present a detailed spatial 'picture' without having to rely upon sounds which are more iconic/symbolic in nature.

9 Appendix

9.1 External information resources and programs used in the development of VAEBuilder

During the development of VAEBuilder the World Wide Web has proved an invaluable resource for information on the various technologies needed to make an integrated, working application. The main areas drawn upon during the design of this software were:

- MFC (Microsoft Foundation Classes) – The most prevalent C++ API for developing Windows applications.
- OpenGL 3-D graphics library.
- Windows sockets – needed to implement the TCP link between the client program and the Huron audio server.
- The Spaceware software development kit (Spaceware SDK for short) used to access the Spaceball.

In addition three main third party tools have been used to make the C++ sources more *maintainable* and *accessible*. These are:

- CVS – a version control system which can be used to track changes in a set of source files and allow backtracking in cases where bugs are inadvertently introduced.
- Doxygen – a documentation tool which is able to produce high quality cross-referenced online and printed documentation from commented sources with the aid of special formatting tags included in the comments.
- Python – an object oriented scripting language which is freely available on the internet at no cost. Python is only used to strip various Microsoft-specific constructs from the source files so that Doxygen can successfully parse them and build up the documentation.

Finding suitable information on all of the tools listed above represented a large part of the work carried out and for convenience the most useful URL's found are listed in the following sections.

9.1.1 Doxygen – a tool for automatically documenting C++ code.

Doxygen is a tool for generating documentation from C++ source files. It is able to generate cross-referenced documentation in HTML and LaTeX formats directly from the sources by parsing the files and making a note of the classes, structures, variables, functions and files involved. With the addition of specially formatted comments placed in the source files it can include more detailed information at the appropriate locations in the documentation. Doxygen saves an enormous amount of work when documenting a C++ project. It has the advantage that the documentation is stored in the source files themselves in the form of specially formatted comments, so whenever a change is made to the source code the documentation will always remain in sync with the code, so long as the comments are updated at the same time.

For further information, documentation and the downloadable package itself see the following URL:

- <http://www.stack.nl/~dimitri/doxygen/>

9.1.2 CVS – a system for version control of source files.

CVS is a tool for *version control*. What this means is that it can be made to track the entire history of every file within a project allowing the changes from version to version to be viewed and also allowing previous versions of a file to be recreated with ease. One of its main uses is in enabling bug tracking, since if a particular part of the software package suddenly stops working properly the developer can simply step back through the changes to the relevant files to find out which particular change introduced the bug. For any medium to large scale software project version control is a must.

CVS is available for free download at the following URL:

- <http://download.cyclic.com/pub/cvs-1.10/cvs-1.10.tar.gz>

An online manual for CVS 1.10 is available at the following URL:

- http://www.delorie.com/gnu/docs/cvs/cvs_toc.html

The following URL contains links to both an online version and .pdf and .ps format versions.

- <http://www.loria.fr/cgi-bin/molli/wilma.cgi/doc.847210383.html>

See the following URLs for other information relating to CVS:

- <http://www.loria.fr/cgi-bin/molli/wilma.cgi/doc>
- <http://www.cyclic.com/cvs/info.html>

9.1.3 Python – the object-oriented scripting language

Python is available from the following URL:

- www.python.org

Python is only used to get the C++ sources into a form which Doxygen can interpret. The problem lies in the fact that Visual C++ has many non-standard constructs and relies heavily upon macros. These constructs and macros must be stripped from the source files in order for Doxygen to work properly. This is achieved by opening a command line window in Windows NT, changing directory to the folder in which the VAEBuilders sources reside and typing:

```
python strip.py
```

This leaves the processed source files in the ‘stripped’ folder of the distribution. See section 9.4 for details on generating VAEBuilders’ LaTeX and HTML documentation.

9.1.4 OpenGL for Windows resources

OpenGL is the industry standard API for programming 3-D graphics. It is available as standard on Windows NT. The primary source of detailed reference information for programming with OpenGL under Windows NT is the MSDN library. For introductory information on OpenGL itself (not Windows-specific) see the following URL:

- <http://reality.sgi.com/opengl/>

For more specific information on how to use OpenGL on Windows NT see the OpenGL Win32 tutorial at the following URL:

- <http://ugweb.cs.ualberta.ca/~blaine/opengl32.html>

For examples of how to set up an OpenGL graphics window using MFC see the following URL:

- <http://www.codeguru.com/opengl/oglwrapperclass.shtml> (OpenGL wrapper class)

The OpenGL programming guide online (known as the 'Red Book')

- <http://fly.cc.fer.hr/~unreal/theredbook/>
and in downloadable postscript format
- <http://graphics.eecs.wsu.edu/cpts442/book.html>

The OpenGL reference guide online (known as the 'Blue Book')

- http://www.dgp.utoronto.ca/people/van/courses/opengl/opengl_index_alpha.html

9.1.5 Windows Sockets Programming Resources

The following is a list of resources which were found to be useful when getting to grips with the intricacies of programming Windows sockets.

Winsock development information

- <http://www.sockets.com/>

The Winsock FAQ

- <http://www.cyberport.com/~tangent/programming/winsock/>

A variety of Winsock examples

- <http://users.neca.com/vmis/wsockprg.htm>

Windows Sockets Programming

- <http://www.goodnet.com/~esnible/winsock.html>

Simple Winsock client and server application examples

- <http://www.cyberport.com/~tangent/programming/winsock/examples/basics/basic-client.cpp>
- <http://www.cyberport.com/~tangent/programming/winsock/examples/basics/basic-server.cpp>

9.1.6 Spaceware SDK and the SpaceBall 3-D input device

The Space Ball 3-D input device is interfaced to a software package using the Spaceware SDK. The documentation is available locally at:

- C:\Program Files\Spacetec\SpaceWare\SDK\hlp\Index.html

assuming the SpaceWare SDK was installed in the default location

9.2 Coordinate Systems and Matrix Conventions

There are three separate coordinate systems used in VAEBuilder which are described below. An additional complication is that the SpaceWare SDK math library stores transformation matrices in *row major order* whereas OpenGL expects transformation matrices in *column major order*.

For example the SpaceBall transformation matrix:

$$\begin{vmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{vmatrix}$$

becomes the OpenGL transformation matrix:

$$\begin{vmatrix} m_{11} & m_{21} & m_{31} & m_{41} \\ m_{12} & m_{22} & m_{32} & m_{42} \\ m_{13} & m_{23} & m_{33} & m_{43} \\ m_{14} & m_{24} & m_{34} & m_{44} \end{vmatrix}$$

This means in practice that whilst SpaceBall transformations are stored in the following format:

$$\begin{vmatrix} \dots & \dots & \dots & 0 \\ \dots & \dots & \dots & 0 \\ \dots & \dots & \dots & 0 \\ t_x & t_y & t_z & 1 \end{vmatrix}$$

OpenGL expects a matrix of the following form:

$$\begin{vmatrix} \dots & \dots & \dots & t_x \\ \dots & \dots & \dots & t_y \\ \dots & \dots & \dots & t_z \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

where t_x , t_y , and t_z are the translation values for the x , y and z axes. OpenGL uses what are known as *homogenous coordinates* (x,y,z,w) . The homogeneous coordinate (x,y,z,w) represents the 3-D coordinate $(x/w,y/w,z/w)$ so long as $w \neq 0$. Homogeneous coordinates are explained in Appendix G of the OpenGL Red Book (see section 9.1.4) but one of their advantages is that they allow translations to be coded in the same matrix as other transformations such as rotation and scaling as shown above.

9.2.1 SpaceBall coordinate system

The coordinate system used by the SpaceBall 3-D input device interprets x , y and z in the following way:

+x	right	-x	left
+y	up	-y	down
+z	forwards	-z	backwards

9.2.2 OpenGL coordinate system

The coordinate system used by OpenGL interprets x , y and z in the following way:

+x	right	-x	left
-----------	-------	-----------	------

+y	up	-y	down
+z	backwards	-z	forwards

9.2.3 LakeDSP coordinate system

Just to make life more complicated a third coordinate system exists for the Huron spatial audio server. This interprets *x*, *y* and *z* in the following way:

+x	right	-x	left
+y	forwards	-y	backwards
+z	up	-z	down

9.3 Steps for setting up the Dell Dimension XPS R400

The following instructions are provided in the event that the whole system has to be set up from scratch due to a hardware failure or system crash.

1. Install Microsoft Visual C++
2. Install MSDN library
3. Install Lake Huron 3.0 software
4. Install Huron license (on floppies)
5. Copy 'wavtosim' program off of Huron 2.7 CD into C:\huron30\bin as it is not included on the Huron 3.0 CD.
6. Go to **Start->Settings->Control Panel** and open **System**. Go to the **Environment** tab and select the **PATH** environment variable. Add 'C:\huron30\bin' to the list of paths to search for executables.
7. Install Spaceware SDK and Spaceball drivers on CD which came with the Spaceball.
8. Open the file 'C:\WINNT\system32\drivers\etc\services' and add the following entries:

```
huron    30000/tcp
huron    30000/udp
```

This tells Windows NT that the Huron audio server will listen for communication requests on port 30000. This port number is specified in the Huron documentation.

If you want to update the Class Library Reference documentation when the code is modified you also need to install Doxygen (see section 9.1.1) and Python (see section 9.1.3) and then add the binary executables for both to your path as well. See section for details about how to rebuild the documentation after changes have been made to the source code.

NOTE: Reinstalling everything on a freshly installed Windows NT machine may also involve installing various Microsoft Service Packs. VAEBuilder was successfully developed using Service Pack 3 initially but later on using Service Pack 5. For more information on service packs and the order in which they must be installed with regard to other software consult a system administrator.

9.3.1 Include and library paths in Visual C++

There are a number of environment variables used by Visual C++ to specify the paths to header files and libraries. These must be updated to include the specific VAEBuilder to function. To be more specific the following headers and libraries are required:

- GL/gl.h

- GL/glu.h
- slim.h
- spwmath.h
- Winsock.h

The following include paths must be added to the project settings in order for the Huron and Spaceware headers to be found:

- C:\huron30\include
- C:\Program Files\Spacetec\SpaceWare\SDK\inc
- C:\Program Files\Spacetec\SpaceWare\SDK\lib\slim\inc

Go to **Project -> Settings...** and select the **C/C++** tab. Then select the **Preprocessor** option in the **Category** list. Add the paths given above as a comma separated list in the **Additional include directories** text box.

The following library paths must be added to the project settings in order for the SpaceWare libraries to be found.

- C:\Program Files\Spacetec\SpaceWare\SDK\lib\uapi\bin\siapp\win32i\release\mthread
- C:\Program Files\Spacetec\SpaceWare\SDK\lib\slim\bin\win32i\debug\mthread

Go to **Project -> Settings...** and select the **Link** tab. Then select the **Input** option in the **Category** list. Add the paths given above as a comma separated list in the **Additional library path** text box.

9.4 Rebuilding the documentation for VAEBuilder

Provided you have installed both Doxygen and Python and updated your path environment variable to include the binary executables for both you can rebuild VAEBuilder's documentation in four easy steps:

1. Open a Command Prompt window
2. Change directory to F:\VAEBuilder or wherever you have unpacked the distribution
3. Type 'python strip.py'
4. Type 'doxygen'


Step 3 removes any MFC specific constructs and macros from the source files and step 4 invokes Doxygen which scans the (modified) source files stored in the 'stripped' folder and generates the HTML output. The output is stored in the 'html' folder of the distribution. Doxygen also generates LaTeX format output which can be used to generate a postscript version if you install LaTeX. Many Linux distributions come with LaTeX pre-installed and this may be the easiest route to generating printable documentation but versions are available for Windows NT.


9.5 Brief instructions on using VAEBuilder


When VAEBuilder is first launched it is not connected to the Huron audio server. In order to use VAEBuilder the following steps must be followed:-

1. Open a TCP connection to the Huron by selecting the *Huron->Open Huron Connection* menu option.
2. Start the Huron VRack application. This should be done remotely from VAEBuilder by selecting the *Huron->Start VRack* menu option.
3. Load a Huron Workspace. Once again this should be done remotely from VAEBuilder by selecting the *Huron->Load Huron Workspace* menu option. This brings up a file dialog window.

VAEBuilder will not allow a workspace to be loaded until the VRack application has been started and will also not allow the VAE to start until a workspace has been loaded. This has the unfortunate side effect that the *Start VRack* and *Load Huron Workspace* menu options have to be selected every time VAEBuilder is run, even if the VRack application was not shut down between sessions.

Once these three steps have been carried out there are two ways to start and stop the VAE. You can either select the *VAE->Activate VAE* menu option or click on the coloured  icon.

Similarly, to stop the VAE either select the *VAE->Deactivate VAE* menu option or click on the middle grey  icon.

The third toolbar button with the lighter  icon is intended to reset the VAE to its initial state but is not used in the current implementation.

Finally when shutting down VAEBuilder it is advisable to stop the active VAE (described above) and then select the *Huron->Close Huron Connection* menu option. If this last step is not carried out the socket connection to the Huron is often not shut down gracefully, which means that a connection cannot be re-established until the Huron socket application is shut down and restarted. This is only a minor inconvenience however.