C1184787
GEOFF BOWER

INNOVATIONS
FINAL REPORT

INNOVATIONS REPORT – CONTENTS

3 – Introduction.
3 – Aims.
5 – Research.
7 – Implementation.
8 – *OpenML.*
11 – *FMOD.*
13 – *OpenGL.*
15 – Results.
16 – Conclusion.
17 – Appendix 1: User Guide.
18 – Appendix 2: Bibliography.

Introduction

This is a report on my attempt to develop an application that will take multiple sound inputs and convert them into live 3d visuals. The libraries used were OpenGL®, OpenML® and FMOD®. This report covers my initial aims, how I went about trying to achieve those aims, the results I achieved and the conclusions I came to.

Aims

The aim of this project was to develop a piece of software that would create visuals for a live band in real time using the signal input from their instruments to drive the visuals. Having seen bands perform with VJ's I observed they greatly increased the visual impact of the band and also enhanced the music somewhat. However, I have noticed very few VJ's use visuals that synchronise with the music. Their repertoire usually consists of mixing together various clips they have sampled or made themselves. Though this is effective if done well, it can get quite tiresome for the VJ. The visuals lose their impact if they become out of time. Also, if the VJ's library of clips is fairly limited then this also reduces the length of time that the clips are appealing. In this application I hope to create a tool that could be used by VJ's to enhance and expand their visual repertoire.

The main function of my program was to provide abstract 3d visuals that are unique to the music being played. In a real situation, the application would need to be able to blend between different visuals in a number of ways, so as the audience don't lose interest. Most VJ packages provide a number of different transitions for the clips loaded. However, this is something that I feel was beyond my capability within this project. I did aim to create more than one visual and an interface to switch between them. This would be the beginnings of implementing this feature into my program.

A feature I wished my application to have is that new visuals will be easy to implement into the program, as the more effects there are the more visually interesting it will be to watch, and the more the creative the user will be able to be. I did not intend to create a user interface to create the visuals, but to code it as such anyone with the knowledge could easily add more visuals by adding to the code.

The other main feature I needed to implement into my application is the ability to take multiple inputs. I had in mind that different instruments could affect different parts of the visuals. From an artistic point of view, the visual display would be much more representative of the music than any visual that just takes the sound as a whole. I like the parallel between the music forming a whole sound from several instruments and the visuals forming a whole image from several patterns responding to different inputs, namely the music being played.

One other idea I had was that that, as visuals are usually projected onto more than one screen, the program could output different visuals to different projectors. This would allow even greater creativity with respect to visual performance. Possibilities include each musician having their own visuals behind them; visuals moving between screens; it would even be possible to have one giant visual spreading across all the screens.

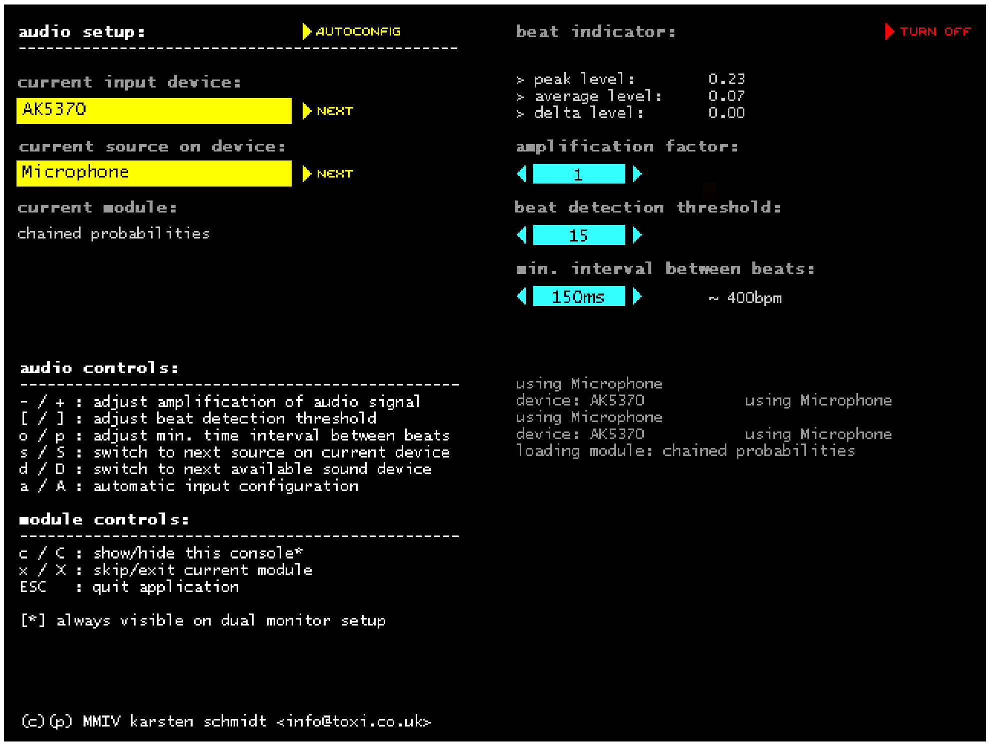
Ultimately, I didn’t have time to produce a piece of software that could perform all of my ideas. My aim was to create a small application that would visualise live music that had the potential to be expanded and developed to accommodate the ideas I had.

Research

My first task was to obtain a number of music visualisation programs to see the kind of effects other people had created. A number of them were plug-ins for popular media players, some were stand alone visualisations. Curiously, some of the stand alone visualisations didn't need you to load in a sound file; it just worked with what was being played on any separate media player. This was a good sign as it suggested sound data could be accessed outside the application creating it, meaning my application could be developed to run alongside other music software programs. For instance, if someone was mixing the band through their laptop using Cubase® the visuals could use the same inputs that Cubase was using.

Nearly all the visualisations I downloaded output 2d visuals, with patterns responding to the wave form. One however, did almost exactly the same as what I wanted my program to do. You would set an input device, and then 3d visuals would respond to the sound input.

The software was called 'The Hub' and I downloaded it from www.toxi.co.uk. This proved to be a very resourceful site. It had many links to interactive visual software as well as programs and code to download. The following image shows the set-up interface of The Hub, and gives a good indication of the sort of controls my program would eventually need.


```
audio setup:                    ▶AUTOCONFIG
----------------------------------------------
current input device:
AK5370                          ▶NEXT
current source on device:
Microphone                      ▶NEXT
current module:
chained probabilities

audio controls:
----------------------------------------------
- / + : adjust amplification of audio signal
[ / ] : adjust beat detection threshold
o / p : adjust min. time interval between beats
s / S : switch to next source on current device
d / D : switch to next available sound device
a / A : automatic input configuration

module controls:
----------------------------------------------
c / C : show/hide this console*
x / X : skip/exit current module
ESC   : quit application

[*] always visible on dual monitor setup

(c)(p) MMIV karsten schmidt <info@toxi.co.uk>
```

```
beat indicator:                          ▶TURN OFF

> peak level:        0.23
> average level:     0.07
> delta level:       0.00
amplification factor:
◀  1  ▶
beat detection threshold:
◀  15  ▶
min. interval between beats:
◀  150ms  ▶        ~ 400bpm

using Microphone
device: AK5370              using Microphone
using Microphone
device: AK5370              using Microphone
loading module: chained probabilities
```

-set up interface for The Hub visualization software, downloaded from www.toxi.co.uk

An interesting feature of this program was all the visuals were external modules, meaning many more could be programmed and added as required. A feature, as mentioned, I was looking to implement somewhat in my own program. For the most part though, these visuals were quite simple; 3d shapes with nice textures expanding in response to, from what I could tell, the volume. Another would draw what looked like 3d shapes made from a revolved section of the waveform. This was quite nice. Many would fly the camera about the scene. If done well this can enhance the impact, but mostly I found it a distraction. Some would jump the camera to a different position in the scene if it detected a beat. Though the change would be in time with the music being played it was most unsettling, as the new camera positions seemed to be random, and each shift felt like a bad jump cut. I would prefer it if the camera moved less and concentrated more on the symmetry of the scene. One effect that did empress me was one of the visuals would launch a number of bright lines into the air upon the start of a sound. Their size, movement and trajectory were influenced throughout the length of the sound. When the sound stopped, the lines would dissipate. Personally, I didn’t intend to write any particle physics into my project, but I liked the effect it had visually and will look into it if continue this project in the future.

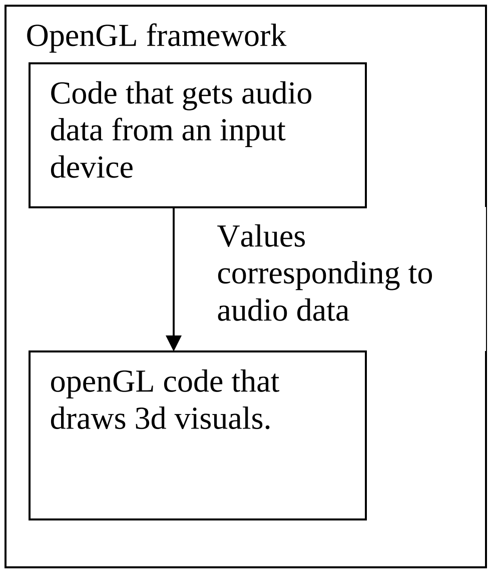
After seeing what could be done, I needed to find out how. I downloaded source code along with some of the visualisations, though it unfortunately didn’t prove very useful. A lot of them were written in languages other than C or C++ which I am unfamiliar with. They also provided little insight into how I could obtain real time input as most of them worked with music files already stored on the computer. I had already decided to use openGL as I wanted 3d visuals, and I had some experience in it from a programming project last year. Most of the visualisations I obtains source code for were all 2d, so again, not much use.

I did download the source code for ‘The Hub’, but was unable to read the files I downloaded. This was quite disappointing as knowing what libraries it used and how the program was structured would have been a really good start to the project. It was probably better that I didn’t get to see it though, as it forced me to develop my own solution to the project and learn more about the API’s I used to code it.

So although the visualisations I found were not much use for technical reference, they did provide much insight into the effects I could create. I realized most of the patterns were only really responding to one input, what looked like the waveform in most cases, the volume level in others. By just repeating this pattern and making it respond in different ways you could make some quite complex visuals. The shape of the music did all the hard work. When the visuals had no input what you had was revolving, flat lines, a lot of the time.

Implementation

To obtain a basic visualization program, the actions it needed to perform, in principle, weren't numerous. I needed three main block of code as I saw it. One handling the sound input, which output values to be fed into another block that had openGL procedures for drawing visuals that respond these values. Then a main block of code that follows the basic openGL structure which controls setting up the inputs and then drawing the visuals (see below).


OpenGL framework
Code that gets audio data from an input device
Values corresponding to audio data
openGL code that draws 3d visuals.

-Diagram to describe basic structure of visualisation program

By openGL framework I mean the basic structure any openGL program follows, which can be summarised thus:

- Initialise openGL (set up drawing modes and window sizes).
- Set openGL display function (function contains drawing code).
- Set openGL reshape function (contains functions to perform on window resizing).
- Set openGL idle function (code to be performed before screen is drawn).
- Set openGL exit function (called on exit).
- Initialise program (contains all code that needs to be called before openGL runs).
- Run openGL main loop (calls an infinite loop in which the previous functions are called).

OpenML

My first task was to get some form of input so I could get some values to draw with. I started with a microphone. After doing some research, I came across a library, OpenML®. The website described it as:

> "…an open source, royalty-free, cross-platform programming environment for capturing, transporting, processing, displaying, and synchronizing digital media - including 2D/3D graphics and audio/video streams."

This sounded perfect. Below is a diagram from the OpenML spec1.0 which was provided in the documentation. From that it seemed clear that I had found the right API. It turned out to be quite a low level API in terms of functionality though, and I found this quite slow and difficult to work with.

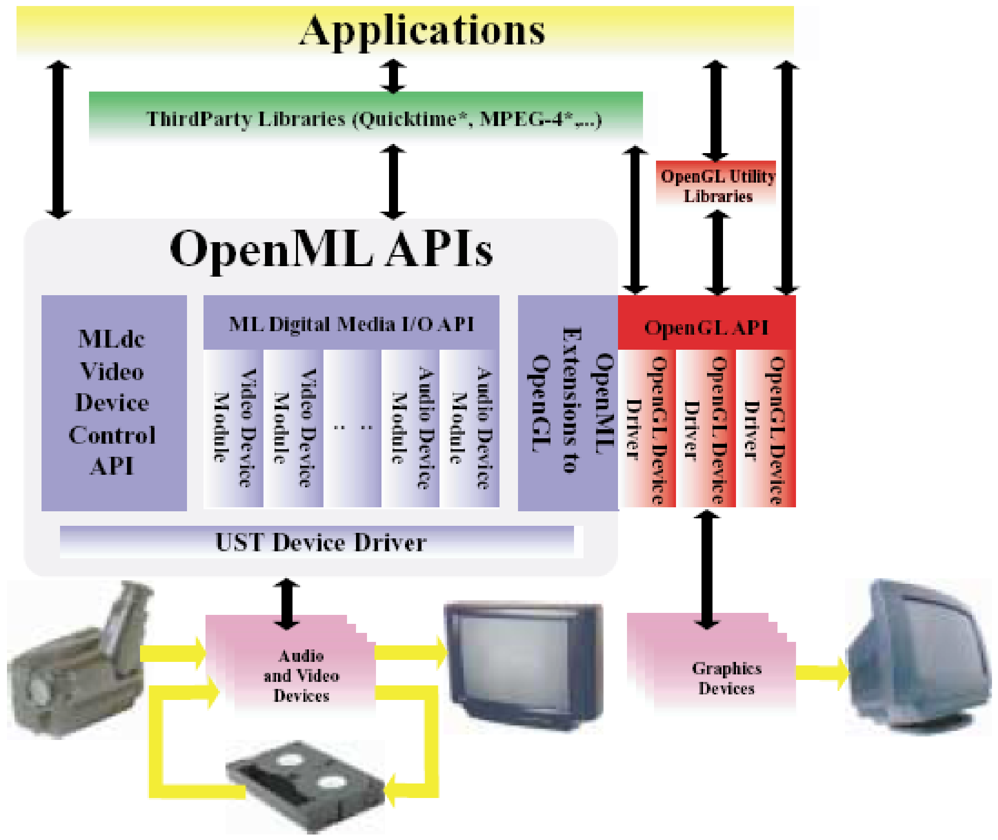


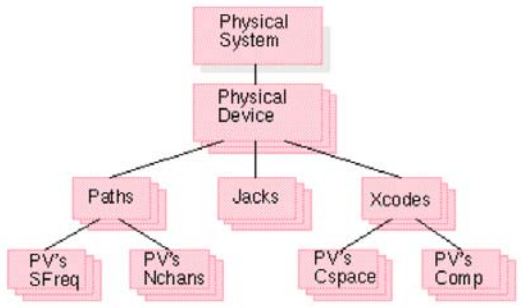
Figure 3.1 The OpenML Programming Environment

- p9 OpenML1.0spec.pdf

OpenML basically worked on what they called the 'capabilities tree'.

> The capabilities tree forms a hierarchy that describes the installed ML devices in the

following order from top to bottom:
1. Physical system
2. Physical devices [e.g. microphones, speakers]
3. Logical devices [e.g. jacks, paths, x-codes, defined below]
4. Supported parameters on the logical devices



-p21-22, MLuserGuide.pdf

Jack- A logical device that is an interface in/out of the system. Examples of jacks are composite video connectors and microphones. Jacks often, but not necessarily, correspond to a physical connector, in
fact, it is possible for a single ML jack to refer to several such connectors. It is also possible for a single physical connector to appear as several logical jacks.

Path- A logical device that provides logical connections between memory and jacks. For example, a video
output path transports data from buffers to a video output jack. Paths are logical entities. Depending on
the device, it is possible for more than one instance of a path to be open and in use at the same time.

Transcoder- A transcoder [x-code] is a logical device that takes data from buffers via an input pipe or pipes, performs an operation on the data, and returns the data to another buffer via an output pipe. The connections from memory to the transcoder, and from the transcoder to memory, are called *pipes*. Example transcoders are DV compression, or JPEG decompression.
-p2 MLuserGuide.pdf

Through this tree you can access any ML device on your system. The process being, in short:

- Get system id;
- Use system id to get available device ids;
- Search a device for desired path, jack or x-code to open;
- Query path, jack or x-code for available options and parameters;
- Query individual parameter characteristics;

I managed to write code that performed all the above and stored the ids in a hierarchy of objects. The first bizarre problem I came across was when I tried to print out this hierarchy to the standard output. For every id you can obtain a name for what that id relates to. If I printed out the name of the id whilst the object was constructing, the printed names were correct. If, however, I used an identical series of loops to when the object was constructing, to print out the names after it had constructed, the output was different. For several id names it output “c-media wave device” which is one of the standard windows sound drivers. Though this output was very strange, the ids still seemed work as they should in later functions, so I decided not to worry about it for the moment.

At this point I was playing around with example code for opening devices and communicating with them. Once a device is open nearly all interfacing is done through messages. A message is defined as an array of a particular structure they call a parameter/value pair (mlPv), defined as such, where the parameter of the last message is ML_END:

```
typedef struct {
     MLint64 param;
     MLvalue value;
     MLint32 length;
     MLint32 maxLength;
} MLpv;
```

-p15 mlUserGuide.pdf

This is the fundamental building block of OpenML. The *param* is a unique numeric identifier for each parameter; and the *value* is a union of several possible types, accommodating for all the possible data types a parameter may have. As an example, the following is a message that would set image width to 1920 and image height to 1080:

```
MLpv controls[3];
controls[0].param = ML_IMAGE_WIDTH_INT32;
controls[0].value.int32 = 1920;
controls[1].param = ML_IMAGE_HEIGHT_INT32;
controls[1].value.int32 = 1080;
controls[2].param = ML_END;
```

-p16 mlUserGuide.pdf

This message would get sent to a device for processing. The device would then return a reply message of the same structure, containing the results of the processing, once processing has finished, and more messages can be sent.

Once a device is open and all parameters set as you wish, the structure of the openML code can be summarized thus:

- Send messages to device, they get queued.
- Get wait handle for asynchronous processing.
- Start path or x-code transfer, queue of messages start to be processed.
- Perform synchronous x-code work.
- Check on the status of the queue.
- Process reply messages.
- Find specific returned parameters to results of processing.
- Repeat as necessary.
- Stop the transfer and close all logical connections.

Opening a device to get some audio data was one step away, except all the example code assumed you had already assigned an audio buffer. There didn't seem to be a structure within openML for them, and I found little information on how to create my own. I had been working with openML for a week now with little positive result. Upon seeking advice on openML or any other method for obtaining microphone input, I was pointed in the direction of FMOD, and some example code where the program played back input from any audio input device.

FMOD

With some manipulation this code was exactly what I needed. I had achieved my initial aim of getting microphone input. The process FMOD went through to set it up showed similarities with OpenML, but used less functions. It can be summarized thus:

- Select output method (Direct Sound, Windows Multimedia Waveout or no sound)
- Select output driver.
- Select mixer
- Initialise FMOD.
- Select input driver (e.g. microphone. Can be set before initialization)
- Allocate buffer to record to.
- Make sample buffer looping and start recording.
- Play back sound

Once I had the FMOD code working I realised another anomaly in my OpenML code. The three output methods provided by FMOD were the first three devices beneath the system in the capabilities tree. For what should have been the Direct Sound driver it printed 'null x-code'. This suggests OpenML wasn't supporting this sound driver. As it is the systems main sound driver this would have made working in OpenML even more difficult.

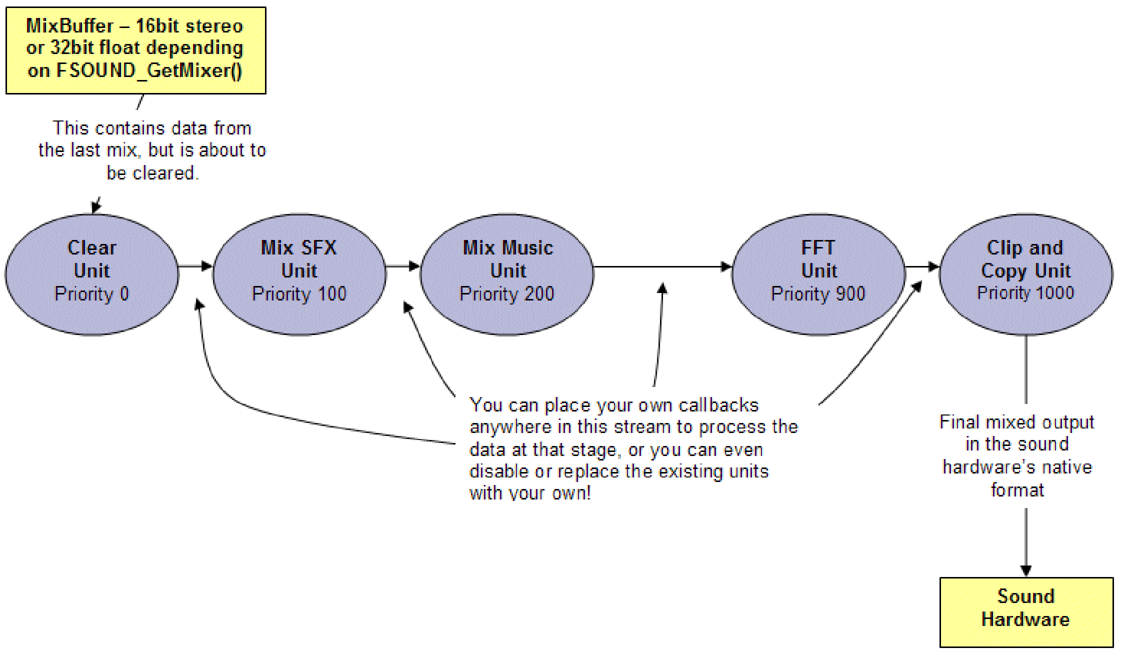
FMOD provides a number of functions to return data about the stream being played, which made getting the values for the visuals very simple. The first value I obtained was the levels, though a function:

```
FSOUND_GetCurrentLevels(channel, &l, &r);
```

By manipulating the left and right input values I got a smoothed value that reperesented the overall VU (by smoothed I mean the input values have been calculated so the output value does not change as erratically as the original, raw input data.)

The other feature of FMOD I used was its ability to perform frequency spectrum analysis. The spectrum analysis unit is an FMOD system DSP unit. The user guide describes it thus:

> The DSP system drives FMOD's software engine.
> Roughly every 25 milliseconds (platform dependant), FMOD mixes a batch of data to be sent to the sound device.
> To do this it executes a chain of algorithms to produce the sound. This chain is executed every time and performs the various jobs such as mixing, clipping and calling user callbacks to process the sound data.
> *-FMOD compiled html help file*


MixBuffer – 16bit stereo or 32bit float depending on FSOUND_GetMixer()
This contains data from the last mix, but is about to be cleared.
Clear Unit Priority 0
Mix SFX Unit Priority 100
Mix Music Unit Priority 200
FFT Unit Priority 900
Clip and Copy Unit Priority 1000
You can place your own callbacks anywhere in this stream to process the data at that stage, or you can even disable or replace the existing units with your own!
Final mixed output in the sound hardware's native format
Sound Hardware

-Diagram describing structure of FMODs DSP system
-FMOD compiled html helfile.

Through the DSP system you can filter the sound data. The help file suggests you could add you own reverb, distortion, EQ or anything else involving sound data, including plotting it for an oscilloscope effect.

The spectrum analysis returns an array of 512 floats, from 1hz to half the sample rate given to FSOUND_init() (so with a standard 44khz output it would produce results up to 22khz). These values represent the different frequencies that make up the incoming sound (in essence, high notes effect the higher frequencies and low notes effect the lower frequencies). Values above 11khz (when initialized at 44khz) become very small however, and consequently graphically uninteresting. For the purposes of this project I only needed the first 256 values. Even then, it's still the first half that's most active, many of my effects ended up only using the first 128 values.

As mentioned, you can program a DSP unit to return values to plot an oscilloscope. I attempted this, using code in one of the FMOD examples as a guide. In principle, it's a simple matter, as raw sound data is simply an array of values that describe the shape of the waveform. Unfortunately, I could not obtain this data. The code the example used was obscure, as it was designed to plot the shape of the oscilloscope to a bitmap image. I simply wanted a sequence of y values. Although as far as I could see my code followed the same structure for setting up and running the DSP unit, but I could only ever get a single value for what I thought should be an array describing the amplitude of the waveform.

OpenGL

After successfully obtaining microphone input, and getting the program to output the levels and spectrum analysis values to the console, the next step was to develop some visuals in openGL that responded to these values. Having some experience from previous projects this did not prove too difficult to set up. The device input is set up in the init() function, before the openGL loop.

In my previous project, I had created objects by loading in *.OBJ files and loading textures onto them. For this project I wanted to explore more how I could get openGL to draw and colour shapes. My program uses some solid objects and some alpha blended, transparent objects. To create a solid material, the following code could be used:

```
GLfloat mat_solid[] = { i/64, 0.0, 1-(i/64), 1.0 };
GLfloat mat_zero[] = { 0.0, 0.0, 0.0, 1.0 };

glMaterialfv(GL_FRONT, GL_EMISSION, mat_zero);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_solid);

glutSolidSphere(.1,5,5);
```

-Example openGL code. Draws solid blue ball

Should one wish to create a transparent object, you could use the following:

```
GLfloat mat_transparent[] = { 0.0, 0.0, 0.8, 0.6 };
GLfloat mat_emission[] = { 0.0, 0.0, 0.3, 0.6 };

glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent);

glEnable (GL_BLEND);
glDepthMask (GL_FALSE);

glBlendFunc (GL_SRC_ALPHA, GL_ONE);
glutSolidSphere(1,20,20);

glDepthMask (GL_TRUE);
glDisable (GL_BLEND);
```

-Example openGL code. Draws transparent blue ball

One other feature I used of openGL that I hadn't before was fog. This helped give some depth to the visual I was creating. This turned out to be really simple to implement. Within the init() function I called:

```
glEnable(GL_FOG);

//fog colour
float fogColour[4] = {0.0, 0.0, 0.0, 1};
glFogfv(GL_FOG_COLOR, fogColour);

//fog blend factor
glFogi(GL_FOG_MODE, GL_EXP);

glFogf(GL_FOG_DENSITY, 0.05);

glHint(GL_FOG_HINT, GL_DONT_CARE);
```

-Example openGL code. Turns on fog

Making the actual visuals was just a case of creatively using the values obtained from FMOD to draw patterns to the screen with various openGL objects. The main parameters affected were position and scale.

Once I had the input and some visuals working, I thought it would be a simple step to make FMOD record from multiple inputs. I repeated the code that set the device to record from and the function that started recording data to memory, but it would not receive any input from a device unless only one had been specified. Unfortunately, I did not have time to investigate this problem further. At this point, my project had developed as far as it could in terms of the time I had left to complete it.

Results

The final application I produced could create two different visuals, switch between the two, but they only responded to one input. More visuals could be written by anyone with basic openGL knowledge, but the code is not flexible enough yet that other parts of it would have to be modified to accommodate for the extra visuals. With respect to my original aims, this project was only a partial success. I realise now that some of these aims were far more ambitious than I had originally anticipated. I didn't even go anywhere near multiple screen output (though I believe this would be possible with a great enough knowledge of OpenML).

The fact I did not solve the multiple input problem was the most disappointing aspect of this project for me. This was one of the more innovative aspects of the project. It was also the key to it being used as a tool to represent the musicians of a band visually. I am not sure if it is even possible to record multiple inputs at once with FMOD. This is definitely something that OpenML could perform, but then I would lose the functionality that FMOD has for retrieving data about the sound.

The best bit of this project is just using it. It's great to play around with the shapes with your voice, seeing how you can make it change. The visuals respond particularly well to actual instruments. The many harmonic tones that overlay the sound of a musical instrument serve to create the most interesting patterns.

I do believe this code has the potential to be developed into the kind of application I imagined in the beginning. I have discovered the tools I need; I just need to learn how to use them better.

Conclusion

Although my final program doesn’t have all the functionality I had envisioned it to have, I am still pleased with the final result. I have learned a lot about sound and how it can be processed on a computer. I expanded my knowledge of openGL. The result is still really fun and looks good. If it is possible to record multiple inputs with FMOD, it will be a short step to getting different parts of a visual to respond to the different sounds. Using the spectrum analysis, it would even be possible to program a game of ‘sound pong’, where the pitch of your voice controls the height of the bat on the screen!

I think the most important lesson I came away with was that there’s a lot more to be learned from actually developing the software, than looking into how other people achieved similar results. If I had started coding sooner I would most likely have learned a lot more. As always with coding:

> There are known knowns. These are things we know that we know. There are known unknowns. That is to say, there are things that we know we don't know. But there are also unknown unknowns. There are things we don't know we don't know.
> *-Donald Rumsfeld*

And it’s the immeasurable number of unknown unknowns that constantly crop up with each new problem that serve to be the most difficult problems to solve, yet they ultimately provide the most useful knowledge.

Appendix 1: User Guide.

- The first thing that will happen when you run the program is you will be asked to select the output type. Direct Sound, the first option, is usually best.
- Next will be listed the direct sound drivers. Pressing ‘1’ will select the default driver, press another number if there is another driver you wish to load.
- Next will be a list of all the input devices available on the system. Select the one you wish to use. Again, ‘1’ will select the default.
- The next call asks if you wish to get another input device. DO NOT press ‘D’ to add another, the code that relates to this doesn’t work. Press any other key instead, and the first visual will load.
- Make noises on your input device and see how the visuals react.
- Press ‘w’ to see the other visual, ‘q’ to go back to the original.

Note: I believe to successfully run the program an up to date copy of Microsoft Visual C++ needs to be installed. This can be downloaded free at http://msdn.microsoft.com/vstudio/express/visualc/default.aspx

Appendix 2: Bibliography

Websites:

http://www.sulaco.co.za/nitrogen/projects.html
http://www.3ps.co.uk/3ps/mysql_manual/index.php?path=http://www.opengl.org/
http://www.toxi.co.uk/
http://us.metamath.org/mpegif/mmmusic.html
http://www.fmod.org/docs/tutorials/basics.htm
http://www.imaja.com/change/tonecolor/
http://www.tstex.com/
http://www.fraktus.com/exo/exo.php
http://www.glassner.com/andrew/cg/research/shapesynth/shapesynth.htm
http://www.khronos.org/openml/
http://techpubs.sgi.com/library/tpl/cgi-bin/browse.cgi?coll=0650&db=bks&cmd=toc&pth=/SGI_Developer/MLSDK_BG
http://www.cs.cornell.edu/dali/
http://www.cescg.org/CESCG-2000/OKubelka/
http://www.opengl.org/resources/code/basics/redbook/index.html

Pdfs and other media:

Supplied with OpenML:
mlUserGuide.pdf
OpenML1.0Spec.pdf
Mlu_manPages .pdf
ml-dev-manual.pdf

FMOD compiled HTML file.

Accellerated C++: Practical programming by example
- Andrew Koenig, Barbara E. Moo. Published by Addison Wesley, 2000.