Title

Research and development into the interactive visualisation of music in a game environment.

Introduction

After research into the visual representation of music, and into interactivity and game design, I have developed a fully playable interactive game engine that incorporates audio information to effect the appearance and game-play of the game. In this report I propose to explore and explain my findings, including the challenges involved, the project's context in regard to other work in similar fields, going on to describe how I have taken what I have learnt in research to develop the engine, and how this is implemented from the technical side. The game engine I have developed is just one possible outcome that has come from my research, but other systems could have been created using a similar approach.

I will later discuss the degree of success of the project, mainly relying on other people's opinions, as I feel this is the only way to test if the project has achieved its goals. This will lead to a discussion of additional elements that could extend the project in the future.

Aim of the project

My intentions were to explore the interaction between player and music in a gaming environment. This includes investigation into what makes a game fun, and how can music be visualised and interacted with successfully in this environment.

By using this research I planned to produce a playable and fun ‘driving’ style game whose tracks are defined, in real-time, by the music supplied to, and where the music is affected by the way the player plays the games, both intentionally and non-intentionally. The focus was on the physics of the engine, but with possibilities to be customisable and extended, and have the ability to add on more complex visuals, be having sufficient output data to build on. I planned to show that the visuals reflected elements of the music (as well as the interactive elements of the track) but not take this much further (e.g. attractive particle effects).

Challenges and targets

The following challenges are those that I foresaw, either unavoidable (due to my aims) or targets that I had imposed on myself that I felt the end result should achieve.

- To create a game that was fun to play. Also, a dynamically created game may be harder to make fun when compared to one that been intricately designed beforehand.
- The difficulty level needs to be predictable and consistent, with the ability to be altered.
- Choosing what and how to extract relevant elements from input music.
- To make the game seem to be a good representation of the music (readable by the user and somewhat predictable).
- Offer interaction between the player and the music – how the music can effect and be effected by the game-play in an understandable way.
- The game needs to have predictable patterns and outputs, so the player can understand what is going on.
- Usable and fun with any music that is input; adaptable. However, it should play differently depending on the music.
- Aesthetic needs to be readable and familiar to users but remaining simple. Focus is not on aesthetic.
- An element of repeatability. The game should be the same when it is loaded with the same music and settings.
- To design and program the game with all this in mind.

Other Work

I have yet to find any system quite like the one I have developed here, but there have been obvious influences from a variety of fields that tie together in my research. These include video games, different forms of music visualisation, and other research and systems based on interactive music visualisation using a computer.

Games:

Vib Ribbon (NanaOn-Sha, 1999)

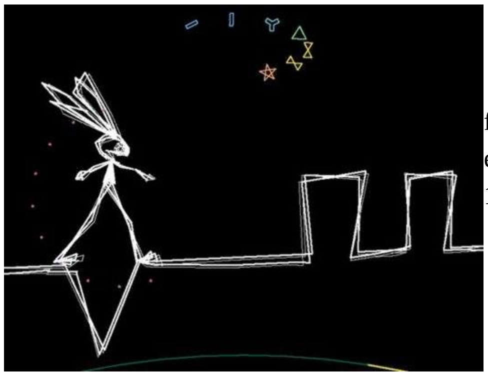

figure 1. Demonstration of how the shape of the level is effected by the music input in Vib Ribbon (NanaOn-Sha, 1999).

Vib Ribbon is a puzzle/platform style of game that uses music input to generate levels. It is based on a very simple idea, and the visuals are minimal. The level is loosely shaped based on the music, by selecting a pattern from four unique elements (see figure 1). The simplicity of the game is a main reason that it is enjoyable, and is something I have considered in my game.

PaRappa the Rapper (SCEA, 1997) and Dance Dance Revolution (Konami, 1998)

In games like ParRappa the Rapper and DDR the player must perform tasks in time with the music and visual cues. In PaRappa the Rapper this is performed by using a control pad, but in DDR you have to perform dance moves on a special mat as input. The gameplay is obviously majorly influenced by the music. However, the music and levels are all predefined, so it is easier for the game designers to have more control over the result. DDR obviously offers an innovative way to control the game.

Wipeout3 (Psygnosis Limited, 1999)

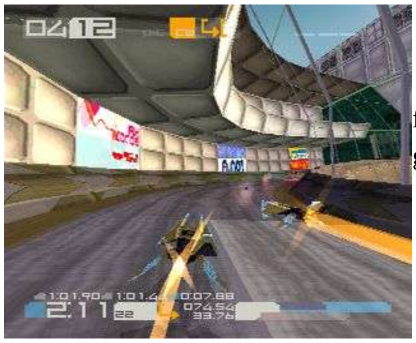
figure 2. The track on Wipeout3 played an influence in my own game Wipeout3 (Psygnosis Limited, 1999)

Wipeout is a fast paced racing game that uses music to enhance the feel but does not bear any direct connection with the track. Although the music chosen is well suited to the level it is not driven by it. The idea of the 'track' that the players race on has influenced the way my own game is designed (figure 2).

Wave Race: Blue Storm (Nintendo, 2001)

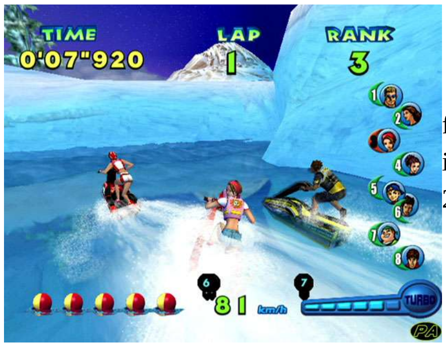

figure 3. Jet-ski style physics like those in Wave Race have been incorporated into my engine. Wave race: Blue Storm (Nintendo, 2001).

Games such as Wave Race employ the sort of physics I have explored in my engine – jet skis over water (see figure 3), because I feel that music lends itself well to be represented as waves, much like the water; although my idea is presented with much more abstract visuals. The aim of my project was to incorporate the effect of bouncing over and cutting through the 'waves' generated by the music.

Outrun (AM2, 1991)

figure 4.Gameplay from old-style racing games have influenced my own project, OutRun (AM2, 1991).

Another influence for the gameplay side of my project is old style racing games such as Outrun. This offers a very distinct feeling semi 3d environment. This is due to the track appearing in front of you as you drive, while the car remains static. Although I use slightly more advanced 3d routines, the fundamental (including the idea of only displaying the section of the track that the player is racing on) ideas are shared between these old console games and my own system.

<u>Music visualization systems:</u>

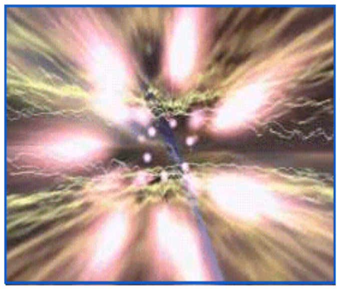
figure 5. Visualisation plugins for media players offer good ideas for how music can be represented in real-time, Nullsoft Winamp plugin - Geiss 4.23 (Geiss, 2003)

Computer audio players e.g. Winamp (Nullsoft) , often include the ability to display advanced music visualizations (see figure 5). This makes for good research into ideas that others have tried in the way of how music can be visually represented. The visuals are generated in real-time to suit whatever music they have as input, much like my project.

Fantasia (Disney, 1940) represents early attempt to visualize musical elements. Some of this is abstract but it is mostly it is more narrative and character based. Although it is good to research to see some visually attractive methods for representing music, Fantasia was obviously meticulously planned many hours spend on the design, which is of course the opposite to creating visuals in real-time.

<u>Other research papers:</u>

I have found some useful work already done in the field of interactive music visualization systems. I'll will describe below some elements that I feel most contributed to my own work.

Interactive Visualisation of Musical Form using VRML by Graves, Hand, & Hugill (1997) discusses their work on creating a system to visualise music input in real-time using various visual shapes and patterns. Their work focuses on Midi music input, which, due its nature of simply being a set of instructions to be implemented by a synthesiser, can undergo more advanced analysis far easier than raw sound input (beat, pitch, instrument, notes etc are all easily accessible). However, most people's favourite music is stored in the latter form, i.e. an actual recording, so thats the route I've decided to take. The paper talks of getting “macro-level (structural) and micro-level” information from the music. Macro-level information includes patterns in the music over a certain time, e.g beat, or increases in pitch, whereas micro-level is more about the input right now. Micro-level is easier to analyse in raw sound data. The paper mentions “Benektine cyber-space”, which is a system where objects can have more than three dimensions by using their x, y, z position and also other dimensions such as size, shape, and colour. These can by mapped to various elements of the music. The visuals that represent the musical elements are referred to as “Visual Metaphors”, which I feel is very appropriate. Apparently Benektine warns that due to the lack of true 3d in a computer display, you have to be careful when mapping sound to the scale of the objects, so that it doesn't interfere with the appearance of depth, e.g an object that gets bigger may appear to be approaching the viewer.

Interactive music visualization (Kubelka) describes the use of 3 main units for the visualisation system that was worked on, the sound analyser, the visualisation module, and a 'scene editor' which is their method for customizing the visualisations. This seems an obvious way to develop the system and is similar to the method I have used, with my program divided up loosely into the sound input and analyser section, a section for the physics and interactivity of the actual game, and the display routines.

Music-Driven Motion Editing: Local Motion Transformations Guided By Music Analysis by Cardle, Barthe, Brooks, & Robinson (2002) describes their system for animating curves using the musical input. They also use midi files as input. However, both the midi score and the analogue sound that is generated from it are analysed. The midi score is used to get a lot of important information regarding to the volume, structure of the music etc. More complex information is then gathered from the synthesised output, such as audio spectrum analysis. It doesn't suggest how this information was extracted from the music, but this isn't what I am interested in investigating with this project, I will be focusing on how this information can be interpreted. Their system uses predefined animation curves, which are affected by the sound input. This is similar to the requirements of my project, as the game needs some structural integrity (the virtual racing track will always need to look like a track and be drivable) and be controlled enough to always make the game fun.

Bringing it together

After my initial research I went on to develop my end result; a familiar racing style game engine. This is an ongoing work-in-progress, and over the course of the project I have been adding and reworking parts of it, after careful play-testing. More technical information about how the engine works can be found in the next next section and accompanying source code, but below is a discussion of the system in its current state.

Visuals:

The visuals are simple, developed from displaying straight lines that approach you, on a plain black background. Although simple (and the initial idea was to develop on these more given the time), I think the visuals lend themselves to the nature of the project quite well, maintaining an abstract and uncluttered feel. To the initial game template I added more details, e.g random corners, then turned the lines into waves to visualise music. I think the visuals represent a familiar metaphor, as we see this sort of frequency analysis visualisation a lot (computer media players and even some home HiFi's). The player character has stayed quite simple as just a sphere, but displays the audio input by changing its size based on the volume. Possible confusion with distance and scale have not been a problem as the size changes so rapidly, and the user is familiar with the common game design rule that dictates that the sphere should stay the same distance from the camera.

Music:

Elements that are taken into account from the music include volume and frequency spectrum analysis. Visually the volume affects the player 'blob' by changing its size. The overall volume is also analysed when automatically adjusting the shape of the track.

The frequency analysis is used extensively in the game as the basis for the shape of the surface of the track.

My initial idea was that the music playing would by unaffected by the game (it would play as if played through a normal media player) but to add to the fun and interactivity of the game I decided to experiment with how the music could be affected by the way the player plays the game. This led me to exploring effects such as slow down and speed up in the music, and mixing different music. The various incorporations of music into the game was an ongoing process with the project, and of course many further developments could be made, as mentioned later.

It is up to the user (and encouraged) that they use their own choice of music to play the game with. The intention of the project was always to be playable with any sort of musical input that it is offered. This means that the game will be specific to the individual, and adaptable to their interests and feelings. Different music can lead to a very different feeling when playing the game, e.g. classical to dance music. This should, I hope, enhance the enjoyment of the game, and promote an interest in testing different tracks to see how they will work.

Gameplay:

The gameplay has a familiar old style semi 3d racing game feel to it. This was partly on purpose, and it is mainly due to the fact that the track constantly approaches the mostly static player. This was a decision to enhance the feeling of the music continuously playing and the player rushing through it. The first trials I did involved the player traveling at a constant velocity in time with the music, but i found it added more interactivity and fun by having the speed of both the player and the music also effected by how the player plays. The music influences the gameplay by effecting the shape of the track surface. I have taken the idea of jet-skis on water into account when designing the physics of the game engine, such that the player skips and bounces on and off of the track surface. When the player is above the surface then they are unable to steer until making contact with the surface again. It was hard to find a balance between providing enough control to the player, and still having the shape of the track (and therefore the music) effecting the gameplay. This balance, and the rest of the gameplay elements were improved and fine tuned by continuously play testing (by myself and others) and adjusting of the controls and the influence of the music.

After the initial play-tests proved to be successful I added more advanced features. This included the ability to be able to switch the music playing at various points in the 'race'. This added an element of the user having control over the game, and being able to play it how *they* wanted to. The book 'A Theory of Fun for Game Design' (Koster, 2005), has been a useful read when considering aspects of gameplay. Koster writes about the importance of the player being able to recognize patterns in the game, which I have tried to incorporate into the ways the tracks are generated (there are no sudden changes in the 'rules' of the game for example). He also mentions that there should be variety and new 'data' for the brain to respond to, while still operating within these patterns. This is apparent in my own game in the way that the game can be played with different music.

As mentioned, the music is effected by the way the user plays the game. This can be intentional and unintentional. For example, if the player struggles to stay in the centre of the track (as is the aim) then the music and the player will slow down. Although at present the game does not have any reward system for driving fast, most players naturally want to drive as fast as possible, so these slow downs are unintended by the user. If the user decides to switch the track to another song, then this is mostly intentional, and promotes a sense of control on the user.

During play, the volume of the music is constantly monitored, and the height of the 'waves' adjusted to ensure a more predictable level of difficulty, and to make sure the game is playable independent of the volume of the music input.

How its done

For those interested in the more technical side of the project; actually how I have taken what I have learnt an implemented into programming the engine, the following section (and the accompanying source code) documents most of the more important elements.

The engine is programmed in C, as a constant work-in-progress. I chose C as it is easier to approach in a 'sketch' like way, with less focus on the need to plan. I could rough out and experiment with bits of code as I worked and redo sections as needed. As the focus of the project was not on programming complicated sound processing functions, I made use of the Fmod sound library, which was very adequate for the functionality I needed. I used Opengl for the graphics, and Glut for the interface. This made it possible to create my project portable from Linux to Windows, and probably other platforms. What follows is a detailed look into the different aspects of the program.

Data structures

Below are the most important data that is stored. As the program is still relatively small, and the prototype-way i worked on the project, many of the variables are global.

Camera structure – used to store information about the virtual camera, i.e. position and viewing angle.

Line item, and line array – The 'track' is stored simply as an array of lines. This array is only 20 elements in size (stored as a type of ring-buffer or queue); only the current section of track being raced on is in memory at any one time, and is continuously being generated in real-time as the player advances. The information stored in the line item structure includes colour, it's angle relative to the line before (which is the basis for the corners in the track), and its shape (both the 'wave' pattern, and its degree of 'roundness').

Player variables – the players position and velocity is stored. Also, a position history is stored, holding the last 10 locations. This is used for a ghosted trial effect on the display.

Input flags – used in the program to know whether a key is being pressed, i.e. for left or right.

Setting up

The program accepts a command line argument when it is run, that becomes the global difficulty setting throughout the game. This effects the speed and also the distance that the player must stay to the centre of the track. Next to be set up is the Glut window and interface, giving a surface for to draw to, and callback functions for receiving input and redrawing. The last to be initialised is the music, which loads in the music files and starts playing them, just as the program initialises the line array and enters the main loop.

Game loop

The main loop is controlled using a timer interrupt to refresh the game state every fraction of a second. This time is dependant on the 'game speed' variable – which takes into account the difficulty setting of the game, i.e the harder the game, more frequently the loop is called, and the faster the game-play.

Firstly in the main loop the player character position is updated. If the track is rotating (if the necessary flag is set) then this is continued. The track rotation refers to when the track is a tube shape and the player travels all the way onto the underside. When this event is encountered, a 'track rotating' flag is set. Each time the game loop is now run the track will rotate a bit more either clockwise or anticlockwise. As this happens the volume level for the alternate track is increased, as the original music volume is decreased. After a full 180 degree rotation, another function is called, which sets up track ready to continue.

Next in the main loop, the offset is adjusted; this makes the track move towards the player, i.e. each loop the whole track is shifted slightly more towards the player, depending on the speed of the player. The track is just a small (20 elements) array of lines and as the closest line reaches the player character, it is destroyed and a new line is added furthest from the player (like a ring-buffer/queue structure).

The new line function adds another line element to the queue, by overwriting the current first item. It is then assigned a random colour. Next it is assigned a shape (the wave pattern) by making a call to the music functions, as described later. When a new line is added a few other less related processes take place. The wave multiplier variable is updated, based on the inverse of the current volume, which controls the height of the 'waves' in each line. This is used to ensure the the average wave height remains approximately the same throughout the track (irrespective of the volume of the input music). The 'shape' of the track is updated. There are various states that the track can be in, regarding its shape. This shape is stored globally and can either be FLAT, ROUND, ROUNDING, FLATTENING, or FLATTENING_WAITING. FLAT is the normal track shape when it is not a 'tube' shape (ROUND). When the shape is set to ROUNDING on FLATTENING, this means that each new line being generated will be incrementally more or less 'round' than the last, respectively, until complete, when its shape will be finally set to its goal, ROUND or FLAT. FLATTENING_WAITING is an extra value which means wait until an appropriate time before being set to FLATTENING. A clearer idea of this can be seen in figure 6.

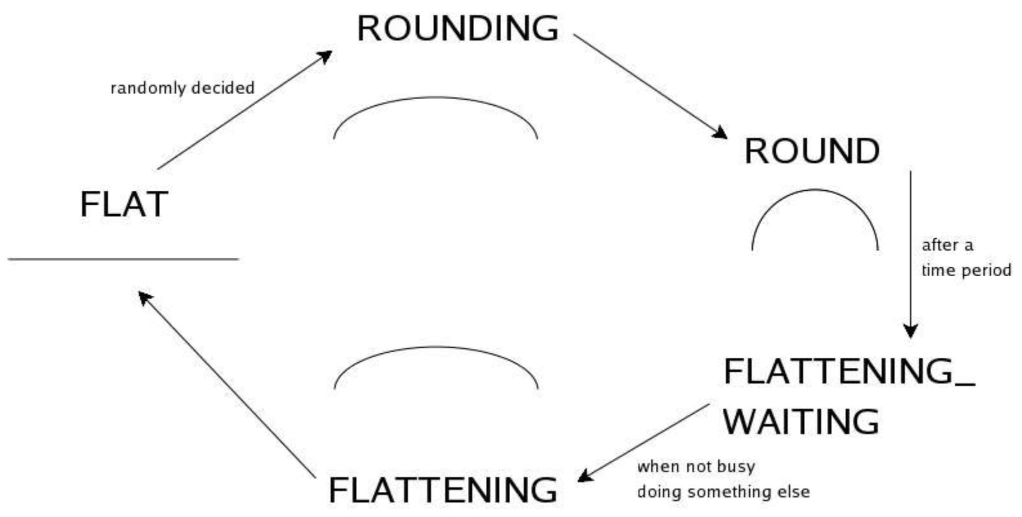

figure 6. Showing the various states the 'shape' of the track can be in.

The track shape value at the time a line is created is stored in that line item, so it can be displayed as necessary when processed. The decision whether to change the normal 'flat' track to the tube shaped 'round' is randomly selected every time a line is added. If it is decided that the track is to change to round, the 'shape' flag is set to ROUNDING. This shape will update with every new line until it is ROUND, then a timer will be activated (for an interupt in 5 secs). When the timer function is called the shape is set back to FLATTENING_WAITING, where it waits to begin its change back to flat. The FLATTENING_WAITING shape will be changed to FLATTENING on the next game loop, unless another event is taking place – specifically the rotation of the track (when the track is flipping upside down), as described later.

The line is also given an angle, which is set globally. There is however, a smoothing value which stops the angles suddenly changing. This is based on the volume of the music at the current time; louder music leads to more violent changes in the angle.

Another thing that takes place when a new line is added is that a new vertical velocity is calculated for the player. This is based in their position in comparison to the wave height at that point; if they are under the wave then the up velocity is increased, if they are 'above' the wave, then it is decreased (ultimately into negative values). This keeps the player 'bobbing' on the surface.

The player has an angle which is the amount they are steering left or right. The main loop checks the input flags and updates this angle, dependant on whether the user is pressing left or right, or nothing. The players position is then updated depending on this angle and their speed. If the player approaches the edge of the track, their speed, and the speed of the music is slowed, unless the track currently round, in which case the track rotating flag is set.

There is also a corner update timer, which is currently hard-coded to interrupt every second. This sets a new global random angle for the track. This value is the angle that every new line that is added to the end of the queue will have. i.e. there will be a new corner angle every second in the game.

Sound Functions

Most of the audio processing was done using the Fmod library, but I wrote my own functions to incorporate these into my program in the necessary ways. For example, I have written functions to extract the frequency spectrum into a format usable by the program, and more advanced methods for setting and retrieving the volume levels.

Draw functions

All the drawing that takes place to the screen involves using OpenGL with GLUT. On each refresh of the screen, a number of things must be redrawn, using the data that is updated by the main loop. I encountered an early problem with the timing being unpredictable in the game. It turned out that I had unwisely included some code that should have been in the main game loop into the redraw loop instead. This meant that the code was sometimes being executed more often than needed. Therefore from then on, I insured the the routines for redrawing to the screen were kept entirely separate from the rest of the game loop, as it should be.

Each time the redisplay function is called, the track and the player must be redrawn.

The track:
By using the array of line items, the entire track can be drawn. The program loops through each line in the array, and draws it in the appropriate place. i.e. firstly, the offset is used to dictate how far away the first line should be drawn from the player. The first line is then drawn, in the colour specified. The waves in the line are stored as values in a separate array for each line, and the y position of each point in the line is offset by the height of the wave at that point. As for the shape of the line (the 'roundness'), sine and cosine functions are used to add the required curvature. The angle of the line is used to rotate line the desired amount. This angle is added to by each line that is drawn, so a corner in the track is created. Each line in the array is processed and drawn this way, by adding a small distance to each line before it is drawn, so that the track appears to continue into the distance.

The player:
The player character is simply a sphere to indicate where on the track the player is. It is simply supplied with the position and size (based on the current volume), then drawn to the screen. To make this more visually attractive, and to help show where the player is, a trial effect is also drawn, by using the array of previous player locations.

Outcome

By first laying down the foundations of the program then building upon this in a trial and error way, adding extra element's, and refining the gameplay, I think I have come up with a game engine that fulfils most of my aims quite successfully. From the moment the program was first in a usable state, I was pleased that the result was already working better than I'd hoped. I consistently tried the engine with different sorts of music to ensure that it worked just as well. Problems such as differing volume levels between songs have been mostly overcome, by automatically adjusting the surface of the track accordingly. This ensures that the difficulty level for the game is consistent (although the game runs differently) independent of the music being played. I managed to incorporate some appropriate features from the music, in a way that I felt was a readable visualisation. Obviously though, the best way to test many of my aims for success was to give the program to others to try, and leaving it up to them to test out their own music as they felt fit. I got a good response from all those who tried it; most people really enjoyed the idea and playing game. This indicates to me that the game offers a good standard of interaction between the user and the music, and the 'rules' and aesthetics of the game are understandable and readable (which I think is the main benefit of its simplicity), which all leads to a fun game. Although the general opinion was that there were still bits that could be added, I took this as a good indication that my project has been successful; it was never supposed to be a finished game, but people like the basic engine enough to see it has good potential, and want to see more.

Future work

Some initial ideas I had to represent the music (beat detection etc) led me into a path of complicated music algorithms, which wasn't the plan of the project. I stuck with the more than ample array of functions that Fmod supplied and focussed on how these could be used. I found your brain makes up a lot of association with the music, even with those elements that have been programmed to be random. However, if I decide to develop the project further in the future, their are a number of things I would like to implement. Currently, there are a few things that are generated by a random algorithm. For example, the angle of the corners. I feel this would be better implemented by using the music to drive these values. This would allow for a more obvious difference between different music, and would better represent the music. Also, one of my aims was that a specific piece of music would have the ability to recreate the same track each time it was used, which is not possible in my implementation. This would add more to a complete game, as players could have another go at a track that they previously 'failed' on. Overall, I feel I could implement more features from the music into the game, for example the beat and stereo volume levels. However, these elements would involve more complicated algorithms to extract.

The engine I have produced has made for a good prototype, and using C has made it easier to play and adapt without a plan, and bits of code here and there. However, if I were to redo or rework the project I would probably do so in a higher level language like C++, now that I have a plan to work to. Now the program is getting larger, this would allow for more structure, and an easier resource to reuse sections of code for other projects.

The libraries I have used have ensured that the program is portable, so their is the future possibility of porting it to other hand-held devices or phones etc., which i think would work well due to the simplicity of the idea, and the audience this would be targeting is a very appropriate one. There may be the need to find alternative sound and graphics libraries if were not already on the target platform, but I would now feel more confident in writing my own for the basic requirements I have here.

The racing game idea that I have created is just a start, and one possible outcome from my research; using my new knowledge on what makes games fun, elements of music, and how the brain interprets music, other games or interactive music systems could be explored with a similar approach, and my research has indicated that there is a relatively limited number of other attempts into similar systems.

Conclusion

Throughout this project I have researched into the many forms of music visualisation and relevant games already developed. From this many varied ideas from been extracted to form my own ideas that have been incorporated into the game engine I have developed. The system has been developed by continuously tested and reworking elements as a constant work-in-progress. My result is a basic engine that looks and feels quite simple, but with a proven appeal and enjoyment level, and which fulfils most of my aims. I have found that for a fun system, simplicity is often the key, aesthetically and in regards to gameplay. Overall the main focus of the project has been completed, but this could obviously be worked on to produce a more advanced and complete system. I have developed new ways of thinking about interactive music, and further realise the potential for many new developments that could be made in the field.

References

Raph Koster (2005) A Theory of Fun for Game Design, Paraglyph Press, Arizona, USA.

Alan Graves, Chris Hand, Andrew Hugill (1997) Interactive Visualisation of Musical Form using VRML, (http://citeseer.ist.psu.edu/graves97interactive.html, accessed 09/03/2006)

Ondrej Kubelka, Interactive music visualization (http://www.cescg.org/CESCG-2000/OKubelka/, accessed 09/03/2006)

Marc Cardle, Loic Barthe, Stephen Brooks, Peter Robinson (2002) Music-Driven Motion Editing: Local Motion Transformations Guided By Music Analysis, p. 38, The 20th Eurographics UK Conference.
(http://doi.ieeecomputersociety.org/10.1109/EGUK.2002.1011270 accessed 09/03/2006)

Images:

figure 1: Vib Ribbon, developed by NanaOn-Sha, published by SCEI (1999)
figure 2: Wipeout3, developed by Psygnosis Ltd, published by SCEE (1999)
figure 3: Wave Race: Blue Storm, Nintendo (2001)
figure 4: Outrun, developed by AM2, published by Sega of America, Inc. (1991)
figure 5: Geiss 4.23, a Nullsoft Winamp plugin by Ryan Geiss (01/02/2003)

Others:

PaRappa the Rapper, SCEA (1997)
Dance Dance Revolution, Konami (1998)
Nullsoft Winamp (http://www.winamp.com)
Fantasia, Walt Disney (1940)
Fmod sound libraries (fmod.org)