Music as an Input Device

Adrian Johnson and SK Semwal
Department of Computer Science, University of Colorado at Colorado Springs
adrianj@uccs.edu | semwal@cs.uccs.edu

Abstract

The entertainment industry uses many methods by which to create visual displays which are synchronized to audio, but few of them are dynamic, real-time solutions. Aspiring musicians[1] may have great interest in ways to create eye candy for personal use and for live shows and also visualizations which assist in the music production process. These graphical representations of the music may be based on a stereo audio signal or musical instrument digital interface data (MIDI data). Several methods of generating openGL visuals that respond to music were created. Three visualizers based on these methods were developed, each suited for a different application. One visualizer was targeted towards music enthusiasts, one for studio engineers, and one for individuals orchestrating live projected visual displays for concerts. These visualizers could also be used with augmented reality outdoor applications using a Nomad augmented reality display system.

1. Motivation

Gaver [1] has introduced the concept of connecting everyday sound with specific actions to provide a metaphor to which the user can attach meaning. The use of sound has also been extended to ubiquitous environments in [2] and various styles are analyzed in [3]. Our motivation in this paper is to use music as an input device to create dynamic, artistic and pleasant animation sequences which meaningfully interpret the music piece. The extension to outdoor environments using an augmented reality system is also proposed.

2. The Initial Visualizer

The first task was to process audio data so that we could use audio phenomena to manipulate openGL objects. The goal was to concentrate on the visual front end of the project more than the audio analysis back end, so the plug-in architecture of Apple's iTunes audio jukebox program was used. iTunes calculates the volume magnitude of each frequency band for both the left and right audio channels. The number of frequency bands the audio is broken down into can be set by a parameter in the plug-in.

With the frequency data available a representation of the audio's EQ was drawn. OpenGL primitives drew a series of narrow, vertical quads across a window. The center of the window represents where the left and right audio channels meet. As the quads are drawn further from the window's horizontal center the frequency is gradually lower. The highest frequencies are represented where the two audio channels meet. With each quad corresponding to a frequency band in one of the channels, the color and brightness of the quad are varied according to the volume of the frequency being represented.

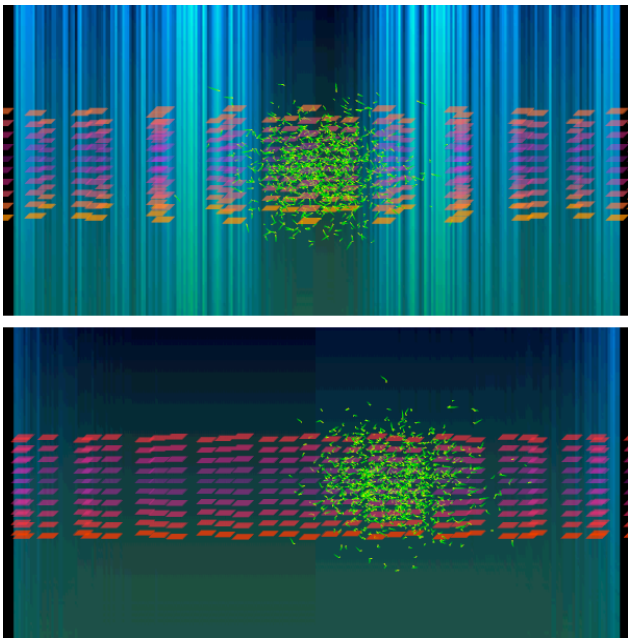

Figure 1. Primitives manipulated by audio

[1] See Adrian Johnson's web page at www.rustcycle.com for details.

By drawing the vertical rectangles based on the raw frequency data one could observe the incoming data that further openGL primitives would be based upon. The vertical quads can be seen as a backdrop for additional primitives in Figure 1. This was very useful in determining whether various functions were operating as intended. A function was created for summing the lower frequencies to capture what was essentially the bass beat in the audio. That value was used to translate members of a second array of quads along the Z axis. The columns of quads were alternatively translated towards and away from the camera. The stronger the bass beat, the further the quads were translated from their resting positions. In an absence of low frequency activity, the quads remained at their original coordinates. To further accentuate the impact of the bass beats, stronger low frequency activity caused more green to be added to the quads' color, making the red quads more orange in appearance.

To make the pan of a track more detectable, the frequency bands were summed for both audio channels and those two values were compared. A particle system was generated to swarm towards the side of the window associated with the dominance in the pan. Essentially, if the track were to pan hard to the left, the random goals of particles would have an offset applied to make the particles travel to the left side of the window. The particles are rendered as line primitives, but like the bass-driven quads, they too could be replaced with complex objects.

3. A Visualizer as a Music Production Tool

The first visualizer plug-in was pleasing to look at and proved that various 3D phenomena can be triggered with audio data. A second plug-in was created which was really much simpler in its processing of the audio, but was more useful for observing the audio's EQ over time. The second plug-in was intended to be a studio tool for fine-tuning the tonal palette of a track. Figure 2 shows two examples of graphics generated by the second iTunes plug-in. This visualizer graphs the frequency volumes with the frequencies being represented at different horizontal locations, similar to the arrangement of vertical bars which display the raw frequency data. As audio is processed by iTunes, the older frequency data is moved away from the camera along the Z axis. It appears as though the incoming audio is in the foreground and the oldest audio is fading into the background. Sounds that incorporate broad frequency ranges appear as bumps in the graph, while sounds that have some type of EQ filter applied to them appear as bumps in a very tight range of frequencies. The rows of tightly grouped lines almost look like a rendering of terrain which varies to the music. By being able to see where instruments disturb the "terrain" it becomes apparent where there may be a need to fill in the mix of the track and also where different instruments may be interfering with each other. To make the display more interesting, camera angles and the color of graph were varied over time.

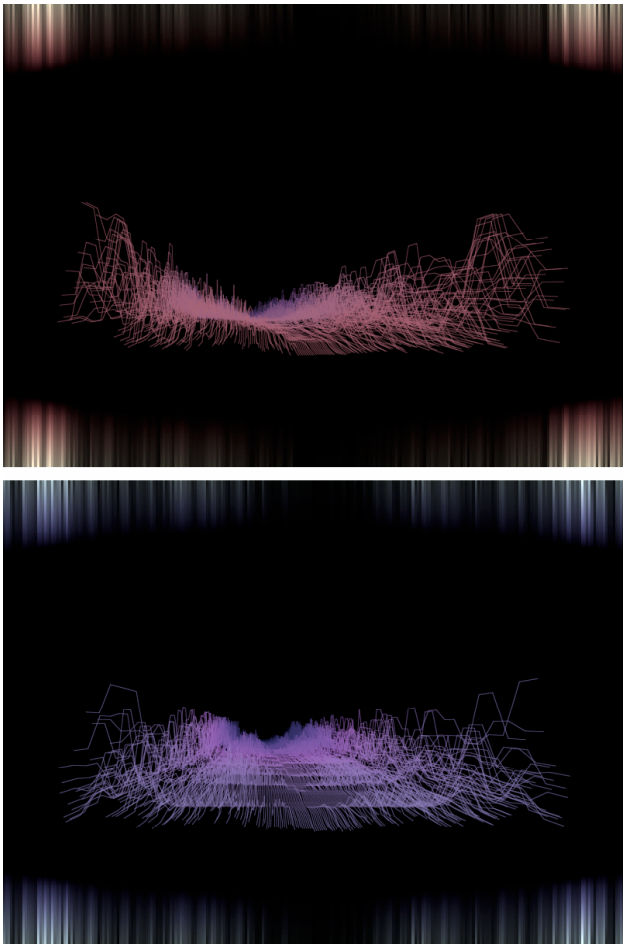

Figure 2. Frequency data renderings

The idea behind the second plug-in will be most useful if audio analysis tools could handle multiple stereo audio files in parallel. In such a scenario, each instrument in a song could have its own audio file and the openGL engine would not be a plug-in for iTunes but rather a component of an audio analysis program. The data from each file would be represented by a different row of openGL lines, with each instrument color-coded so that it would be easier to see where frequency overlaps occur. We intend to look at the plug-in architecture of several music production suites to determine if it's feasible to create graphics with a plug-in. Most music authoring tools have plug-ins for audio manipulation, and may not allow for creation of visuals. It would be very useful to write a plug-in that could assist an individual within the production environment, instead of requiring the musician to export the instruments' audio as separate files.

4. Precisely Triggering Behaviors via MIDI

In the first iTunes plug-in low frequencies were summed to create approximations of where bass beats occured. If MIDI data was examined, that data could contain information where not only bass beats occur, but where notes for other instruments occur. If MIDI data was used to trigger openGL events, we could trigger events with a much finer level of control. MIDI data for most music is not available readily, and it was not our intention to synch openGL to music created by other people. We wanted to be able to use MIDI data from compositions to trigger manipulation of 3D objects and for a MIDI signal to be used as an input device.

In the music production suite, compositions can be written which have MIDI instructions for multiple instruments. Each instrument's notes, pan instructions, volume instructions, and other information is carried by a MIDI channel. In one of the song documents created by the first author, a new MIDI channel was created for giving instructions to an openGL visualizer engine, rather than for sending instructions to a synthesizer module. Within this channel, selections of notes from instruments were copied from the song, and then these notes were altered. For example, to only have the openGL engine perform an object manipulation on the first and third beats in a 4/4 composition having a beat on every count, it was just a matter of deleting the MIDI events representing notes on the second and fourth beats. After the notes were placed at the desired periodicity, the MIDI events were moved to a single note value. For the initial openGL behavior, which was a tinting of the camera's view, the program was listening for one specific note. As new behaviors were added, each different behavior was to only be triggered by its specific corresponding note.

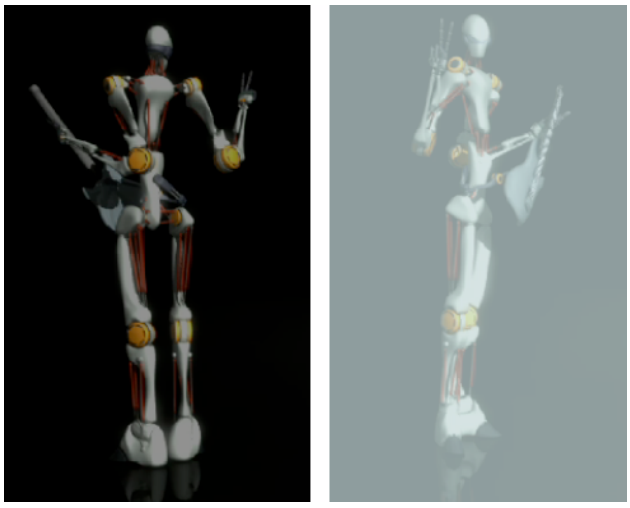

Figure 3. Manipulating QuickTime textures

QuickTime movie files were mapped to a quad as a texture and set to be the focus of the visualizer. The first behavior, the tinting of the openGL view, was done by placing a quad near the camera and manipulating the color and alpha values. Additional behaviors were more involved. With the QT files set to loop, a behavior was created to pause the video and then fast-forward a set number of frames upon receiving the MIDI signal. This behavior was slightly modified to have the QT fast-forward to selected frames picked because they appeared interesting when shown in series. When the trigger was received it caused the movie's rate of animation to increase in order to quickly reach the target frame, and upon reaching the target frame it would then pause again until the next trigger. A third behavior was to cause the textured quad to pulsate along the Z axis upon trigger. The quad was quickly moved towards and away from the camera in order to make the image appear to enlarge and then return to normal. In Figure 3 an animation of a virtual guitarist character created in LightWave 3D was used as texture data and the movie was advanced and tinted in the right frame.

Using video from natural phenomena also works well in this visualizer. Some real world scenes have been filmed by the first author, and plans are to eventually film all of the following: water droplets striking a water surface, colored liquids mixing, fire, trees swaying in the wind, foliage from the vantage point of a moving vehicle, and the sun setting behind nearby mountains. The warping of time will also produce interesting effects with video of urban traffic (night and day shots), people dancing, people engaged in athletics, and crowds of people in urban areas.

5. Conclusions and Future Work

These three openGL visualizers allowed for music to be used to control 3D objects in different manners and for different applications. Manipulation of the 3D objects can have utility, as with the second visualizer, but can also be for pure entertainment value, as with the other two programs. Many artists' web sites have a collection of "special features" items such as a behind the scenes look in the studio, alternate mixes of songs, and perhaps a look into the musicians' past. An entertaining iTunes plug-in could be a unique bonus item for fans of a musical artist. Such a plug-in could also have "secret" graphic displays which are only shown when the artist's music is being played.

Plans are to develop MIDI based implementations as they can be controlled with the most precision and it allows for the graphics rendering to be done on a separate machine from the machine which orchestrates the audio. The quads could have footage of the same phenomena, with each quad having video from a different camera

angle. The multiple camera movie data could just as easily come from several virtual cameras in a 3D animation package such as Maya or LightWave 3D. Plans are to also explore the creation of openGL terrain, water, and other complex geometries for manipulation via MIDI.

The visualizers presented in this paper can also be used in conjunction with an augmented reality device, such as the Nomad display system (Figure 4) available in our Wearable Computing Laboratory. The Nomad system delivers information directly to the user by superimposing digital images on their field of vision, creating a see-through 17-inch display which seems to be floating at arm's length everywhere the individual looks. The Nomad system allows monocular graphics and images to be superimposed on live outdoor action. The visualizer output from a notebook could be directly displayed on the Nomad. In the future, smaller and more fashionable augmented reality devices will certainly make this concept more practical.

Figure 4. Nomad augmented reality system

Visualizers are used far beyond the desktop; they're often used on center stage. With projected visual displays at concerts being more prevalent each year, it seems desirable to have graphics which closely mimic the music in order to pull the audience further into the concert experience. In the best case scenario members of the audience will witness a sensory synesthesia. While an artist could show video footage which ties in well conceptually with the music being performed, by using a tool such as the third visualizer presented in this paper, the artist can manipulate the video rhythmically and heighten the connection between the media. Though a visualizer engine can function based on pre-programmed MIDI events, by using keyboards and other MIDI instruments a musician can dynamically alter video and entire 3D environments in real-time. With old stage tricks becoming passé, technology driven touches will give some musicians an edge over the competition.

Selected References

[1] WW Gaver, The SonicFinder: An interface that used auditory icons, Human Computer Interaction, vol. 4, pp. 67-94, 1989.

[2] GR Borden IV, An Aural Interface for ubiquitous computing, ISWC2002 proceedings, pp 143-144.

[3] P Resnick and R.A. Virizi, Relief from the audio interface blues: Expanding the spectrum of menu-list and from styles, ACM Transactions on Computer Human Interaction, vol. 2, pp. 145-176, 1995.