Fleece: Graphical 3-D Trajectory Specification Software for Spatialization in Live Performance

Harry D. Castle
Center for Research and Computing and the Arts (CRCA)
University of California at San Diego
La Jolla, California 92093
hcastle@ucsd.edu

Abstract
This article describes the functionality and application of “Fleece,” a program written by the author for the specification and subsequent manipulation in real time of the motion of sound objects in three dimensional space. Fleece was written in C to run under Windows 98/NT on Intel-based machines, as well as under Irix on Silicon Graphics machines. All graphical rendering is done using OpenGL, and the program is designed to communicate with a machine running spatialization software over ethernet, TCP/IP socket connections. The paper consists of 1) a description of program capabilities, 2) aesthetic concerns and goals of the program, and 3) detail of a specific hardware/software configuration as used in an actual performance.

INTRODUCTION
Fleece is a program that I wrote beginning in the summer of 1997. While primarily used thus far as a means of graphically specifying and monitoring the movement of sound sources in two dimensions, the program suggests a wider range of application. The graphical elements of Fleece are implemented using OpenGL, and the flexibility of Fleece is greatly augmented by the incorporation of inter-machine communication capability via sockets.

Fleece does not represent a generalized solution to spatialized sound source trajectory specification, but rather reflects the interest of the author in achieving certain kinds of behavioral results for use in responsive, improvisational environments.

DESCRIPTION OF PROGRAM CAPABILITIES
Overview and Terminology
Fleece allows for the positioning of an arbitrary number of point sources in three dimensional space. Each sound source is represented graphically on the screen as a half-diamond with a vertical stem (a Lozenge). The Lozenge is rendered in a three dimensional viewing volume that can be oriented according to the user’s preferred angle of viewing. Associated with each Lozenge is a list of points describing the path along which the Lozenge will travel (its PanPath). Each Lozenge may be assigned a particular speed at which it travels along its path, a negative speed corresponding to reverse motion along the PanPath.

Associated with each PanPath is, optionally, a set of Breakpoints. Breakpoints indicate a location along the path at which the Lozenge motion may change. Currently implemented possibilities are Stop, Reverse Direction, Change Path, and Ignore. A Breakpoint can also act as an attachment point for another PanPath, allowing for the complex chaining of paths.

Also associated with each Lozenge is a TCP client socket that may be connected to a server socket associated with any application that is interested in the Lozenge’s position. Each time a Lozenge position changes, its current (x,y,z) position is broadcast via this socket. The receiving socket may of course be on the machine that is running Fleece, but the flexibility to distribute data to several machines allows for separate processing of graphics and audio computations for improved performance.

PanPath Specification
A single PanPath has many attributes, but its central defining feature is a list of points. The points are contained in ascii text files with the “.pts” extension. Each file contains a list of (x,y,z) coordinates represented as floating point values, one (x,y,z) coordinate per line. Fleece has a drawing mode that allows for the generation of two-dimensional paths by drawing on a cartesian grid. The drawing is free-hand, and continuous motion as well as point-to-point drawing are both allowed. Although the editing features are rather limited at this time, greater
precision may be achieved by editing the “pts” files conceivably possible to generate or manipulate point lists by any means imaginable. For example, a small C program or csch script may be written to generate the points describing a circle with greater precision than could be achieved manually.

A PanPath is further described by a set of transformational parameters that are applied to it before it is rendered. The transformations describe the PanPath’s current size (scaling), location (translation) and orientation in three-dimensional space. Scaling is specified as a multiplier along each of the three axes, so a circular path centered about the origin would with an X-Scaling factor greater that 1.0 would appear stretched equally along both the positive and negative X-axis. Scaling is applied similarly along the Y and Z axes. Since the distortion of the path by scaling is dependent upon each point’s position relative to the origin, all scaling operations are applied before any translation or orientation transformations.

The orientation for a PanPath is determined by its rotation about each of the three axes, and is specified in terms of an HPR (heading, pitch, roll) set of rotations. The rotations are performed in the order roll, pitch, heading.

The location of the PanPath is finally established by a set of translations along each of the three axes. The current position of the Lozenge is treated as a point along the PanPath and as such undergoes all of the same transformations. Thus in the final rendering, the PanPath appears in three-dimensional space with a particular size, location, and orientation, and the Lozenge appears above it (oriented vertically), pointing to the current position.

In addition to positional information, each PanPath is given a unique color, and may be rendered as points or as connected line segments. The speed and current position for the Lozenge associated with each path are also maintained along with the rest of the PanPath data.

Implementation of a Single PanPath

As a program intended for use in real-time, improvisational settings, Fleece reflects a greater concern with general properties of motion than with moving between two fixed points over a fixed time interval. Thus the lines of motion are referred to here as paths and the Lozenge given a particular speed. This stands in contrast to a more through-composed model which might be expected to specify a list of destination points and an associated list of travel times themselves. By storing the coordinates as ascii text it is between points. The latter type of specification may be achieved through Fleece, but is not a central design issue.

The speed of a Lozenge is computed as a change in distance over unit time slice. The unit time slice most often used is 33 milliseconds or roughly the desired time between OpenGL redisplays. Finer time grain may be used, but in applying the current position to spatialization it is often sufficient to let the audio software do the smoothing (as for example with the lin~ object in Max/MSP or sgi-Max)

Thus for each time slice the distance travelled is spread across the line segments connecting the PanPath points, and linear interpolation is used to determine the current position as it falls between points. The advantage of this method is that the Lozenge will move along the path at its designated speed regardless of the density or scarcity of points describing the path: it will move at the same speed around a curve specified by many points as it will along a straightaway specified by as few as two points.

While this is the typical mode of Lozenge movement, an additional “Clamp” mode is also available. In this mode the Lozenge movement is updated internally as usual, but a change in position is only reported when the current position moves beyond one or more points in the list. In this case the point most recently passed is reported as the current location, thus clamping current positions to points supplied in the list and bypassing any interpolation.

PanPath Chaining and Transitions

Multiple sets of PanPath data associated with the same list of points may be sequentially chained to represent multiple orientations, speeds, etc. for that path. Transitional behaviors may then be established to modulate over time between consecutive PanPath parameter sets along the chain. The only currently implemented transition type is linear interpolation over a specified time duration. Thus while the Lozenge travels along a path, the size, location, orientation, and Lozenge speed will vary linearly as one set of values is progressively replaced by the next. In this manner the entire disposition of a path may be animated.

As with the breakpoints, transitions may exhibit one of several behaviors once a transition is completed. The transition may simply stop and maintain the current set of parameters upon completion, or it may continue on to the next set of PanPath parameters in the chain, or it may reverse the process and begin to transition back to the previous set of parameters.
Since the aim of this design is not necessarily to produce a scripted list of settings through which to move, but rather to foster a rich environment for interactive manipulation, the behavior of the transitions is directional. In other words, the speed and mode of interpolation when moving from one set of parameters to the next may be different when moving in the opposite direction. This makes it possible to ping-pong between two different parameter sets, while moving at a different speed, if desired, in one transformational direction than in the other.

While even complex morphing sequences of a path may be fairly easy to grasp visually, the reported movement of the Lozenge along the path is yet further abstracted. It is quite likely that repeated transitions between the same set of parameters will yield a very different path through three-dimensional space for the associated Lozenge. Nevertheless, if it is desirable to constrain the Lozenge to a single static location, it can be achieved by setting all scaling factors to zero, effectively collapsing the entire path to a point at the origin. In this situation, the translation coordinates will dictate the final location of the Lozenge.

Interactive Manipulation of PanPaths

A list of points may be loaded as a file and displayed in the OpenGL viewing window. From that point on a series of keystroke commands adjust the size, location, and orientation of the path, as well as the speed and start/stop status of the Lozenge. All path-specific keyboard commands apply to the "current" PanPath. Other sequences of keystrokes cycle through existing paths; the current path is highlighted for visual identification.

A chain of PanPaths is built by duplicating the current path, and then altering its parameters as described above. An additional variable to set is the forward/reverse transition speed.

Keystroke commands exists for most of the editing and functional operations described in this paper. In order to facilitate control in live performance, Fleece creates a server socket which may be used by any task that wishes to send keystroke equivalent commands. The input handler recognizes an extended set of commands to accommodate continuous controller types of information that are desirable but would not derive from keyboard interaction. To date, Fleece has been interfaced to a WingMan Warrior Joystick and to a Buchla Lightening II.

Socket Functionality

It seemed important from the outset to allow Fleece to communicate positional coordinates to other machines for processing due to the cpu-intensive nature of real-time graphical rendering. Realtime DSP calculations are equally intensive and so it made sense to simply separate the two and avoid graphic/audio cpu contention issues altogether. The natural choice was to communicate via ethernet connection. TCP socket was used rather than UDP because it was reportedly more reliable and guaranteed that packets arrived in the order transmitted. I am aware that subsequent projects have been realized using UDP sockets without reported incident.

TCP sockets are a little fussier about their status of connectivity, among other things, but it was eventually possible to get them to behave almost as flexibly as UDP sockets. Either the server socket application or the client socket application may be started or stopped at any time, and they will nevertheless find each other and re-connect automatically. This is especially useful for development situations in which either the sending or receiving task is repeatedly taken up and down for modifications.

Similar socket functionality has been built into an sgi-Max external object so that a dsp patch can communicate with Fleece. All sockets are non-blocking, and the max external automatically goes into polling mode (a la metro) if it determines that a connection is not active.

AESTHETIC CONCERNS AND GOALS

My overall design goal was to create software with enough inherent layered complexity to produce interesting and perhaps unpredictable results. Thus Fleece encourages the proliferation of sound sources running along complexly varying paths in ways that can create a profound interaction of moving elements around an auditory space. In this way sound sources are imbued with general motion behaviors that may be constrained if desired, but in general are more likely to defy absolute predictability at any given moment.

Although coordinates may be specified with a great deal of precision, Fleece encourages a more hands-on, experimental approach to developing gestures. A path is roughed out, then scaled, positioned, etc. until the general results sound as desired. Similarly, the inherent lack of control over exact repeatability orients attention towards the characteristics of gesture rather than refined orchestration of precision events.

The graphical display of Fleece, although somewhat engaging, is not of significant interest in terms of content that it may add to a work, and as a result has never been projected during performance. As avenues of interactivity develop, however, the graphical display may gain in relevance.
FLEECE IN LIVE PERFORMANCE

“End Game”

Fleece has been used in several performance situations, most notably in collaborations between percussionist Vanessa Tomlinson and myself. A piece of ours entitled “End Game” has been performed in various venues and demonstrates a particular application of Fleece in live performance.

Throughout the piece, Tomlinson bows a cymbal, continually varying the timbral richness and spectral content of the sound in sympathy to its overall presence in the listening space. The sound from the cymbal is processed through delays and pitch shifters, and the results are distributed through a four channel speaker system. The processing level is progressively increased throughout the piece, although it remains fairly subtle to the end. The panning activity is fast yet distant in the opening of the piece, and slower, lush, and encompassing by the end.

Configuration

All sound processing was achieved using a Silicon Graphics O2 running Max. Microphone input from the cymbal was taken into the O2, processed, and output via a Yamaha 02R mixer to a four channel system. Fleece ran on a 233Mhz Intel-based machine under NT. Both computers were connected to a local area network and communication was established via TCP socket connection.

The Max patch had two main sections: an effect processing section and a quadraphonic panning section. The effect processing section consisted of two independently running, dynamically varying delay lines (with feedback), each connected to a bank of four pitch shifters. The pitch shifters were actually delays implemented using vd~ (variable delay) which can handle a rapidly varying tap point. When the tap moves rapidly from one setting to the next, a change in pitch is effected.

This section of the patch had many parameters that were automated, and the parameter values varied within ranges that could be controlled during performance. Of particular interest here are the range over which the vd~ tap was allowed to vary, and the speed at which it would change. By controlling these parameters it was possible to influence the depth of the “swoop” of the pitch shifting. We began with very little change in pitch and gradually increased the potential range over the duration of the piece.

The second section of the patch handled the quadraphonic distribution, and contained two Max externals. The first external managed the TCP socket connection and received coordinate data from Fleece. Each packet contained a sound source (Lozenge) id number, followed by the (x,y) coordinates of the source. The second external took the (x,y) coordinates as input and computed the gain for each of the four loudspeakers.

Although the spatialization of sound was rather crude – essentially quad panning with reverberation - a new model for quad panning was devised for this piece. The speakers were assumed to be situated along the circumference of a unit circle with the listener at the center. Since the Lozenges move about the virtual space in somewhat unpredictable ways, it was important to have a model that accounted for a Lozenge passing through the interior of the unit circle. The model used assumes that a sound source at the origin corresponds to unity gain in all speakers. Outside of the circle, the gain diminishes conventionally as one over the square of the distance from the listener.

In “End Game,” Fleece provided the coordinates for three independently moving sound sources. These were applied to each of the two processed audio streams, and to the unprocessed cymbal sound as well. The paths were translated to a position distant from the audience at the beginning of the piece, and gradually moved in closer and closer, at the same time slowed down the speed at which all three sources moved along their paths. The engulfing beauty of the inharmonic spectra from the cymbal, delayed, wobbled, and in constant motion, resulted in a vividly memorable sonic event.

Conclusion

"End Game" is an early and fairly simple example of what can be achieved by incorporating Fleece as a controllable generative element. It makes use of only a small subset of the complex, multidimensional data Fleece can generate. Plans are currently under way for performances that will explore more of the many avenues of inquiry suggested by Fleece's unique model of data and its traversal.