Adding Sound to the XTANGO Animator

Steven Robbins

September, 1995

Technical Report CS-95-13

Abstract. This is the third in a series of technical reports dealing with the use of sound by programs. In this report a modification to the XTANGO Animator to allow sound generation by the Animator is described.

Division of Computer Science
The University of Texas at San Antonio
San Antonio, TX 78249

1 Introduction

The XTANGO animation package [2] is a powerful tool for showing the dynamic behavior of a program. A simplified interface to XTANGO is provided by the `animator`. This program simply reads ASCII text from standard input and interprets each input line as an animation command. Simple commands allow the creation of objects such as rectangles, triangles, circles, or text while other commands move these objects along a straight line path.

I have developed a method of augmenting this animation package to use sound generated by a MIDI device. At this point the supported features include the generation of a tone with a given timber (instrument), pitch, and volume while a text object is moving. An option exists for the generation of another sound when the move is completed. Additionally, the pitch can be modified as the object is moved.

Most of the changes to the package are make to the animator program itself as this is the program that interprets the commands. The animator calls XTANGO functions to perform the animation. The only modification to the XTANGO source is in the file `xtangodraw.c` and the modification involves only 2 lines. A global variable is declared and initialized at the top of this file:

```
void (*sound_fun)(void) = NULL;
```

This is a pointer to a function which is initialized to NULL. A second line is added to the function `TANGO_Text` which is the last function in the `xtangodraw.c` file. Before this two-line function returns the following is inserted:

```
if (sound_fun != NULL) sound_fun();
```

Since `sound_fun` is initialized to NULL, this has no effect unless `sound_fun` has been modified and so the modified XTANGO package can be used in applications that do not support sound. This means that only one XTANGO package needs to be supported on a system, and the sound features are available without adding significant overhead when sound is not used.

`TANGO_Text` is called each time a text object is drawn. Moving an object in TANGO is accomplished by a sequence of operations in which the object is erased and then redrawn at a slightly different position along the path of motion. If the `sound_fun` pointer is not NULL, the function it points to is called each time the text is moved along its path.

The animator is linked with the XTANGO library, and so it can set the `sound_fun` pointer as desired. Commands have been added to the animator to initialize the sound device and to produce sounds as text is moved.

The animator sound interface is easy to use. To produce a sound of constant pitch during a move, just send the animator the command:

```
nextnote n m p 0
```

where n, m, and p are integers representing the pitch, volume, and channel (instrument). The last parameter can be used to change the pitch during the motion of the text. This sound will be used when the next animator `moveto` command is executed.

If the command

```
endnote n m p 0
```

is also sent before the move, then the corresponding sound is produced when the move is complete.

2 The User Interface

This section gives a complete reference manual for the commands which have been added to the animator at the time this report was written.

```
usesound sounddev
```

This command initializes the sound device. Use `/dev/ttyS0` for the first serial port (COM1) under Linux Slackware 2.2 or `/dev/term/a` for the first serial port under Solaris 2. An error message is sent to standard error if an error occurs.

```
initchan channel instrument
```

This command associates the given channel with the given instrument. Channel numbers are between 0 and 15, and instrument numbers are between 0 and 127. See [1] for a list of the instruments that can be used for a general MIDI device such as the Korg X5DR. This device associates channel 9 with the General MIDI drum kit when it is turned on.

```
nextnote pitch velocity channel increment
```

This command causes the animator to play the note with the given pitch and velocity (volume) on the given channel the next time text is moved. The pitch and volume are each between 0 and 127. A volume of 0 represents silence. See [1] for the relationship between the pitch number and the note that is played. The increment is an integer which may be positive, negative, or zero. Each time text is moved a small amount the increment is added to the pitch, and the note is played. This allows the pitch to vary as the text is moved along a line.

```
endnote pitch velocity channel duration
```

This command sets a note which will start when the next text `moveto` completes. The duration must be included in this command but is is not currently used. Instead, the note is turned off by the next `endnote` command when a `moveto` completes.

3 Implementation Details

The animator keeps track of three note objects each of type `notetype`

```
typedef struct {
   int flag;
   int pitch;
   int velocity;
   int channel;
   int duration;
   int pitchinc;
} notetype;
```

These are called `nnote`, `enote`, and `eoffnote`. The `nnote` contains the next note to play. The `nextnote` animator command sets the `pitch`, `velocity`, `channel`, and `pitchinc` members of this variable from its parameters and also sets the `flag` to true. The `nextnote` command also sets the `sound_fun` pointer to point to the `increment_note` function if the `pitchinc` parameter was not zero. When a text move from an animator `moveto` command occurs, `moveto` starts the note if `nnote.flag` is set. If the `nnote.flag` is set when the `moveto` completes, the note is turned off.

If the `pitchinc` was not zero, the `increment_note` function is called each time a small text move takes place. This turns off the previous note stored in `nnote`, increments `nnote.pitch` by `nnote.pitchinc` and then starts the new note. This allows for the change in pitch as the text is moved.

The `endnote` command is used to produce a note when the `moveto` completes. It sets the `pitch`, `channel`, and `duration` members of `enote` from its parameters as well as setting its `enote.flag` to true. Before the `moveto` returns, it starts this note if `enote.flag` is set. The `eoffnote` variable is used to eventually turn off this ending note. The `eoffnote` is set to the old `enote` value when a new `enote` is set by the `endnote` animator command. If this variable has its flag set when a `moveto` completes, the note is stopped. An `endnote` with a velocity of 0 can be used to turn off the previous endnote without initiating a new one.

4 Code Segments

The following represents most of the code that was added to the animator.

```
notetype nnote = {0,0,0,0,0,0};
notetype enote = {0,0,0,0,0,0};
notetype eoffnote = {0,0,0,0,0,0};
```

```
int sfd = -1;

void send_two_bytes(int b1, int b2)
{
   unsigned char playmsg[2];
   playmsg[0]= (unsigned char) b1;
   playmsg[1]= (unsigned char) b2;
   write(sfd,playmsg,2);
}

void send_three_bytes(int b1, int b2, int b3)
{
   unsigned char playmsg[3];
   playmsg[0]= (unsigned char) b1;
   playmsg[1]= (unsigned char) b2;
   playmsg[2]= (unsigned char) b3;
   write(sfd,playmsg,3);
}

void all_notes_off()
{
   send_two_bytes(0xbc,0x7a);
}

void start_note(int pitch, int velocity, int channel)
{
   send_three_bytes(0x90+channel,pitch,velocity);
}

void end_note(int pitch, int velocity, int channel)
{
   send_three_bytes(0x80+channel,pitch,velocity);
}

void increment_note(void)
{
   if (nnote.pitchinc != 0) {
      end_note(nnote.pitch,nnote.velocity,nnote.channel);
      nnote.pitch = nnote.pitch+nnote.pitchinc;
      start_note(nnote.pitch,nnote.velocity,nnote.channel);
   }
}
```

```
void
usesound(str)
   char *str;
{
   char cmd[SLEN];
   char sound_dev[SLEN];
   struct termios mytermio;
   int retval;
   speed_t speed;

   sscanf(str, "%s %s",cmd,sound_dev);
   fprintf(stderr,"opening sound device %s\n",sound_dev);
   sfd = open(sound_dev,O_RDWR);
   if (sfd < 0) {
      fprintf(stderr,"Error opening sound device: %s\n",sound_dev);
      return;
   }
   retval = tcgetattr(sfd,&mytermio);
   if (retval < 0) {
      fprintf(stderr,"Error getting termio structure for fd=%d\n",sfd);
      return;
   }
   speed = cfgetospeed(&mytermio);
   fprintf(stderr,"Old speed paramter is %d\n",(int)speed);
   retval = cfsetospeed(&mytermio,B38400);
   if (retval < 0) {
      fprintf(stderr,"Error setting baud rate in termio structure\n");
      return;
   }
   retval = tcsetattr(sfd,TCSANOW,&mytermio);
   if (retval < 0) {
      fprintf(stderr,
         "Error setting new termio structure of sound output\n");
      return;
   }
   retval = tcgetattr(sfd,&mytermio);
   if (retval < 0) {
      fprintf(stderr,
         "Error getting termio structure for fd=%d second time\n",sfd);
      return;
   }
```

```
   speed = cfgetospeed(&mytermio);
   fprintf(stderr,
      "New speed paramter is %d, B38400=%d\n",(int)speed,(int)B38400);

 }

void
initchan(str)
   char *str;
{
   char cmd[SLEN];
   int chan,instr;

   sscanf(str, "%s %d %d",cmd,&chan,&instr);
   send_three_bytes(0xc0+chan,0,instr);
 }

void
nextnote(str)
   char *str;
{
   char cmd[SLEN];
   int pitch,vel,chan,inc;

   sscanf(str, "%s %d %d %d %d",cmd,&pitch,&vel,&chan,&inc);
   nnote.pitch = pitch;
   nnote.velocity = vel;
   nnote.channel = chan;
   nnote.pitchinc = inc;
   if (vel > 0) nnote.flag = 1;
   else nnote.flag = 0;
   if (inc == 0) sound_fun = NULL;
   else sound_fun = &increment_note;
 }

void
endnote(str)
   char *str;
{
   char cmd[SLEN];
   int pitch,vel,chan,dur;
```

```
   sscanf(str, "%s %d %d %d %d",cmd,&pitch,&vel,&chan,&dur);
   if (enote.flag) eoffnote = enote;
   enote.pitch = pitch;
   enote.velocity = vel;
   enote.channel = chan;
   enote.duration = dur;
   enote.pitchinc = 0;
   if (vel > 0) enote.flag = 1;
   else enote.flag = 0;
 }
```

5 A Library for XTANGO Sound

It is more convenient for programs to call functions than to send output to standard output. We provide a procedural interface to parts of the animator.

The following functions should require little explanation for those who have used the animator.

```
void initialize_sound(char *dev)
{
   (void)printf("usesound %s\n",dev);
   (void)fflush(stdout);
}

void set_sound_channel(int chan, int instr)
{
   (void)printf("initchan %d %d\n",chan,instr);
   (void)fflush(stdout);
}

void set_move_note(int pitch, int volume, int channel, int incr)
{
   (void)printf("nextnote %d %d %d %d\n",pitch,volume,channel,incr);
   (void)fflush(stdout);
}
```

```
void clear_move_note()
{
   (void)printf("nextnote %d %d %d %d\n",0,0,0,0);
   (void)fflush(stdout);
}

void set_end_note(int pitch, int volume, int channel)
{
   (void)printf("endnote %d %d %d 0\n",pitch,volume,channel);
   (void)fflush(stdout);
}

void stop_end_note(int pitch, int volume, int channel)
{
   (void)printf("endnote %d %d %d 0\n",pitch,volume,channel);
   (void)fflush(stdout);
}

void moveto_id(int id1, int id2)
{
   (void)printf("moveto %d %d\n",id1,id2);
   (void)fflush(stdout);
}

void make_text_flex(int id, double x, double y, char *color,
                    char *font, char *str)
{
   (void)printf("flextext %d %3.4f %3.4f 1 %s %s %s\n",
                id,x,y,color,font,str);
   fflush(stdout);
}

void make_circle(int id, double centerx, double centery,
                 double radius, char *color, char *fill)
{
   (void)printf("circle %d %3.4f %3.4f %3.4f %s %s\n",
         id,centerx,centery,radius,color,fill);
}
```

6 An example

The following example uses the Solaris 2 first serial port `/dev/term/a` and sets channel 1 to be the vibes. It creates a text object containing the word `MOVING` with ID 1000 and two circles with IDs 1001 and 1002. It moves the text to the center of the first circle with increasing notes and then to the center of the second circle with decreasing notes. When done, it produces a clash of symbols.

```
#define ID1 1000
#define ID2 1001
#define ID3 1002
#define ID4 1003
                                        /* Initialize the serial port */
   initialize_sound("/dev/term/a");
                           /* Set channel 1 to instrument 11 (vibes) */
   set_sound_channel(1,11);
                                              /* Create three objects */
   make_text_flex(ID1,0.2,0.2,"black","8x16","MOVING");
   make_circle(ID2,0.4,0.8,0.1,"red","outline");
   make_circle(ID3,0.6,0.15,0.1,"blue","outline");
       /* Set note to 60 = C3, volume = 100, channel 1, increment 3 */
   set_move_note(60,100,1,3);
   moveto_id(ID1,ID2);
      /* Set note to 81 = 81, volume = 100, channel 1, increment -3 */
   set_move_note(81,100,1,-3);
          /* Channel 9 is the drum kit and note 8 is a cymbal clash */
   set_end_note(8,100,9);
   moveto_id(ID1,ID3);
   clear_move_note();
   make_text_flex(ID4,0.05,0.05,"purple","5x8","The End");
```

7 The Table

The table given on the next page lists the correspondence between numeric values and the things they can represent. For each number in the range 0 to 127, 5 values are given. The

first is the instrument that is represented by the number when the number is used in a channel initialization command. Next is my perception of how the volume of the note behaves while the note is on. This is called the envelope of the note and is in the column labeled L. I have used three classifications. F means that the note dies out fast. S means that the note dies out slowly. C means that the volume is constant as long as the note is on so that it does not die out until it is stopped. The third entry is the Drum Kit sound that the number corresponds to if it is used as the pitch for a note on channel 9. Only notes 28-87 represent valid Drum Kit sounds. Next is my perception of the envelope of that Drum Kit sound. Last is the note that the number represents when used as the pitch of a start note or stop note command.

I have picked out a few of the instruments and Drum Kit sounds as being an interesting subset to examine when determining what to use for an auralization. These are listed in boldface. The serious user will want to listen to all of the MIDI instruments and drum sounds.

References

[1] S. Robbins, "Controlling the Korg X5DR Synthesizer from a UNIX Program," UTSA Division of Computer Science Technical Report, CS-95-12.

[2] J. T. Stasko, "Animating algorithms with XTANGO," SIGACT News, vol. 23, number 2, pp 67-71, 1992

Num	Instrument	L	Drum	L	Note	Num	Instrument	L	Drum	L	Note
0	**G01 Piano**	**M**			C -2	64	G65 Soprano Sax	C	Open Conga	F	E 3
1	G02 Brite Piano	M			C# -2	65	G66 Alto Sax	C	Hi Timbal	F	F 3
2	G03 Hammer Piano	M			D -2	66	G67 Tenor Sax	C	Lo Timbal	F	F# 3
3	G04 Honky Tonk	M			D# -2	67	G68 Baritone Sax	C	Agogo	F	G 3
4	G05 New Timesh	M			E -2	68	**G69 Sweet Oboe**	**C**	Agogo	F	G# 3
5	G06 Digi Piano	M			F -2	69	G70 English Horn	C	**Cabasa**	**F**	A 3
6	G07 Harpssichord	M			F# -2	70	G71 Bassoon Oboe	C	Maracas	F	A# 3
7	G08 Clavichord	M			G -2	71	**G72 Clarinet**	**C**	**Whistle S**	**F**	B 3
8	**G09 Celesta**	**M**			G# -2	72	**G73 Piccolo**	**C**	**Whistle L**	**M**	C 4
9	G10 Glocken	M			A -2	73	G74 Flute	C	**Guiro S**	**F**	C# 4
10	G11 Music Box	M			A# -2	74	G75 Recorder	C	**Guiro L**	**M**	D 4
11	**G12 Vibes**	**M**			B -2	75	G76 Pan Flute	C	Claves	F	D# 4
12	**G13 Marimba**	**M**			C -1	76	G77 Bottle	C	**WoodBlock2**	**F**	E 4
13	G14 Xylophone	M			C# -1	77	G78 Shakuhachi	C	WoodBlock3	F	F 4
14	G15 Tubular	M			D -1	78	**G79 Whistle**	**C**	**Mute Cuica**	**F**	F# 4
15	G16 Santur	M			D# -1	79	G80 Ocarina	C	**Open Cuica**	**F**	G 4
16	**G17 Full Organ**	**C**			E -1	80	G81 Square Wave	C	MuteTriang	F	G# 4
17	G18 Perc Organ	C			F -1	81	G82 Saw Wave	C	**OpenTriang**	**F**	A 4
18	G19 BX - 3 Organ	C			F# -1	82	G83 Syn Caliope	C	Cabasa	F	A# 4
19	**G20 Church Pipe**	**C**			G -1	83	G84 Syn Chiff	C	JingleBell	M	B 4
20	G21 Positive	C			G# -1	84	G85 Charang	M	Bell Tree	F	C 5
21	G22 Musette	C			A -1	85	G86 Air Chorus	C	Castanet	F	C# 5
22	**G23 Harmonica**	**C**			A# -1	86	**G87 Rezzo 4ths**	**C**	Side Kick	F	D 5
23	G24 Tango	C			B -1	87	G88 Bass & Lead	C	Taiko Lo	F	D# 5
24	**G25 Classic Guitar**	**M**			C -0	88	G89 Fantaasio	C			E 5
25	G26 Acoustic Guitar	M			C# -0	89	G90 Warm Pad	C			F 5
26	G27 Jazz Guitar	M			D -0	90	G91 Poly Pad	C			F# 5
27	G28 Clean Guitar	M			D# -0	91	G92 Hhost Pad	C			G 5
28	G29 Mute Guitar	M	**Rock Kick**	**F**	E -0	92	**G93 Bowed Glass**	**C**			G# 5
29	G30 Over Drive	M	Snare 3	F	F -0	93	G94 Metal Pad	C			A 5
30	G31 Dist Guitar	M	Open HH	F	F# -0	94	G95 Halo Pad	C			A# 5
31	**G32 Rock Monics**	**C**	Fat Kick	F	G -0	95	G96 Sweep	C			B 5
32	G33 Jass Bass	M	Timbales	F	G# 0	96	**G97 Ice Rain**	**M**			C 6
33	G34 Deep Bass	M	Snare 1	F	A 0	97	G98 Sound Track	C			C# 6
34	G35 Pick Bass	M	**RollSnare1**	**M**	A# 0	98	G99 Crystal	M			D 6
35	G36 Fretless	M	Real Kick	F	B 0	99	G100 Atmosphere	C			D# 6
36	G37 Slap Bass 1	M	ProcesKick	F	C 1	100	G101 Brightness	C			E 6
37	G38 Slap Bass 2	M	Side Kick	F	C# 1	101	G102 Goblin	C			F 6
38	G39 Synth Bass 1	M	**Rock Snare**	**F**	D 1	102	G103 Echo Drop	C			F# 6
39	G40 Synth Bass 2	C	**Hand Claps**	**F**	D# 1	103	G104 Star Theme	C			G 6
40	**G41 Violin**	**C**	LightSnare	F	E 1	104	G105 Sitar	M			G# 6
41	G42 Viola	C	Tom Lo	F	F 1	105	G106 Banjo	M			A 6
42	G43 Cello	C	**Tite HH**	**F**	F# 1	106	G107 Shamisen	M			A# 6
43	G44 Contra Bass	C	Tom Lo	F	G 1	107	G108 Koto	M			B 6
44	**G45 Tremolo Strings**	**C**	Pedal HH	F	G# 1	108	**G109 Kalimba**	**M**			C 7
45	**G46 Pizzicato**	**F**	Tom Lo	F	A 1	109	G110 Scotland	C			C# 7
46	G47 Harp	M	Open HH	**F**	A# 1	110	G111 Fiddle	C			D 7
47	**G48 Timpani**	C	Tom Hi	F	B 1	111	G112 Shanai	C			D# 7
48	**G49 Marcato**	**C**	Tom Hi	F	C 2	112	G113 Metal Bell	M			E 7
49	G50 Slow String	C	**Crash Cym**	**M**	C# 2	113	**G114 Agogo**	**M**			F 7
50	**G51 Annalog Pad**	C	**Tom Hi**	F	D 2	114	G115 Steel Drums	M			F# 7
51	G52 String Pad	C	Ride Edge	F	D# 2	115	**G116 Wood Block**	**M**			G 7
52	G53 Choir	C	**China Cym**	**M**	E 2	116	**G117 Taiko**	**M**			G# 7
53	G54 Doo Voice	C	**Ride Cup**	**F**	F 2	117	G118 Tom	M			A 7
54	G55 Voices	C	Tambourine	F	F# 2	118	G119 Synth Tom	M			A# 7
55	**G56 Orch Hot**	**F**	**Splash Cym**	M	**G 2**	119	**G120 Rev Cymbol**	**M**			B 7
56	**G57 Trumpet**	C	**Cowbell**	**F**	G# 2	120	**G121 Fret Noise**	**M**			C 8
57	G58 Trombone	C	Crash Cym	F	A 2	121	G122 Noise Cliff	M			C# 8
58	G59 Tuba	C	**Vibraslap**	**M**	A# 2	122	**G123 Seashore**	**M**			D 8
59	G60 Muted Trumpet	C	Ride Cym 1	M	B 2	123	**G124 Birds**	**C**			D# 8
60	G61 French Horn	C	Hi Bongo	F	C 3	124	**G125 Telephone**	**C**			E 8
61	G62 Brass	C	Lo Bongo	F	C# 3	125	**G126 Helicopter**	**C**			F 8
62	G63 Syn Brass 1	C	Mute Conga	F	D 3	126	**G127 Stadium**	**C**			F# 8
63	G64 Syn Brass 2	C	Open Conga	F	D# 3	127	**G128 Gun Shot**	**M**			G 8