Hooking game audio: Using Pd and OSC for rapid DSP development

Andy Farnell

Date: 11 October 2007

Synopsis

In order to prototype game audio DSP you need to be able to hear the sound design alongside the running game. In this example we will use the Open Sound Control protocol to directly bind to the events in a very simple game. We will then use OSC messages to parameterise some simple sounds.

Why use OSC?

Some games developers like to use a language called Lua because it is a very fast embeddable scripting language with a small footprint that can be used as glue for bindings or "hooks" between different pieces of game code. One reason is that it interfaces nicely with C++ objects. Here we will not use Lua, because the example is so trivial that we will make direct bindings to the game code. This is possibly much more instructive. Instead we will directly use a flexible messaging system known as OSC (Open Sound Control). This is very suitable for games for a number of reasons. Firstly, like Lua it is extremely small and efficient. OSC is really nothing more than a definition, a protocol for sending message data to unique addresses over a socket connection. It uses a URL type address space, for example we might define a sample replay unit called "Engine35" which exists within the object "Truck3" and has some parameters like start, stop and speed. To talk to this sound object we send messages of the form `/Truck3/Engine35/Speed 100` which are routed to the correct object by the URL. Loose strings or floats following the address string are taken as parameters. Secondly, it is designed precisely for driving sound generation objects that are decoupled from controller code, so it has the capacity to carry time-tagged messages that can be delivered with the correct timing and order even if sent over TCP sockets. This means we can run a dedicated DAW for synthesis and sample mixing on a completely different

machine from the one on which the game graphics are run.

Choosing a game.

The game I picked for this example is Xinvaders by Johny Goldman. It is a small, easily understood program that comes in a just a few C source files and will compile fine on anything that can run X-Windows. For those of you who aren't old fogeys, Space Invaders is the original 2D shoot-em-up game from the late 1970s in which you have a movable base ship, some rocks to hide behind while wave upon wave of enemy hordes suicidally attack your base. See this site http://www.spaceinvaders.de/ for more background information. The original game written by Toshihiro Nishikado for Taito, http://www.taito.co.jp/ had some very recognisable sounds, not implemented in Goldmans code, which we can have fun designing in Pd. In fact the simplicity and effectiveness of the Space Invader sounds has been mentioned in studies of game music and sound. From "Strategies for narrative and adaptive game scoring" by Axel Berndt and Knut Hartmann (2007)...

> "A repetitive stepwise descending four-tone sequence illustrates the approaching hostile UFOs. The closer they come the more the increase in speed and difficulty of the game. Likewise, the tempo of the four-tone sequence accelerates and mediates an increasingly suspenseful precipitance."

The game code takes about 30 seconds to compile on a typical machine so we can easily start playing with the program and tweaking the OSC interface.

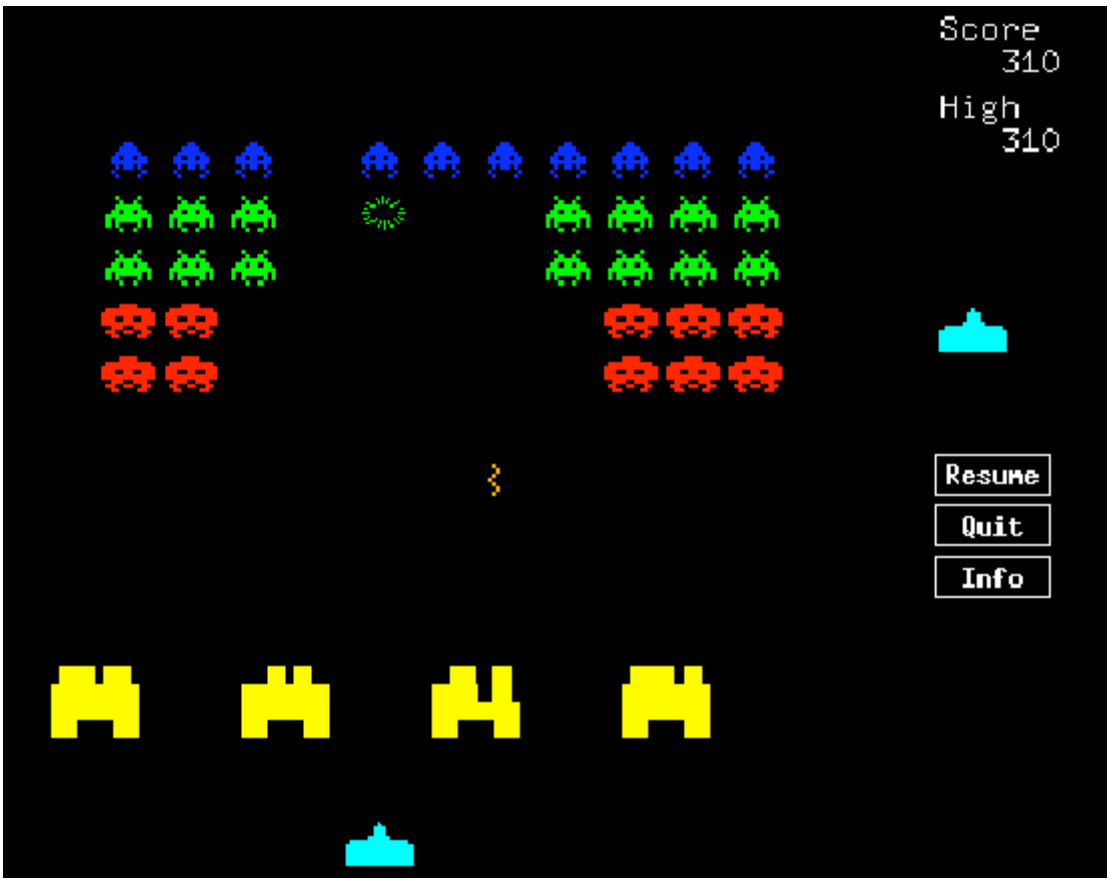

Figure 1: Johny Goldmans "XInvaders"

Compiling the game source with OSC

Download the source package from here

http://www.ibiblio.org/pub/Linux/games/arcade/invaders/xinvaders-2.1.1.tar.gz

Unpack the archive with `gunzip xinvaders-2.1.1.tar.gz; tar -xvf xinvaders-2.1.1.tar` and enter the directory created. Let's take a look at the code. After unpacking the source archive there seems to be 7 C files, some header files and a bunch of graphics bitmaps.

```
$ ls *.c
base.c main.c score.c shot.c spacers.c vaders.c widget.c
```

and the header files

```
$ ls *.h
acconfig.h config.h me.h patchlevel.h vaders.h
```

Type `make` and if all goes well you should be able to start the game by typing `./xinvaders` in the current directory. Movement and shooting are both accomplished

by the mouse. If you don't have a 3 button mouse you might want to tweak the source to use the keyboard instead while doing this exercise.

```
$ make
$ ./xinvaders
```

Next we need to use the OSC library. First grab the OSC code. The one we will use is `liblo` from Steve Harris and Nicholas Humfrey, which is an easy to use and compact pure C implementation that doesn't have a few of the more sophisticated OSC protocol trimmings. Download it from:
http://liblo.sourceforge.net/
Follow the installation instructions to put the library object on your system. Open `main.c` from Xinvaders and add the following line after all the other includes. (On my system `liblo.h` has installed to `/usr/include/lo/lo.h`)

add to main.c

```
#include "lo/lo.h"
```

The next two lines are added to function `main()` so that they are executed as soon as the program is run. The first calls `lo_address_new()` to create an OSC socket on port 7770. The first parameter is the host which defaults to localhost on 127.0.0.1 as a default, or when passed NULL. Then we send a test packet using `lo_send()`, using our port handle `t` then an OSC address `/foo/bar/` and finally our test data which is three strings and a float. These have no meaning at the moment. In reality we might send an initial message from the game to load a patch and initialise the sound system,

add to function main() in main.c

```
int main(Cardinal argc, char **argv){
t = lo_address_new(NULL, "7770");
lo_send(t, "/foo/bar", "ssf", "create", "invaders", 1.0f);
```

One last thing before we can recompile and test the OSC connection, we must make a change to the Makefile so that our new library is linked in. Locate the line beginning with `LIBS =` and make the following changes.

modify the Makefile

```
LIBS = -lXaw -lXmu -lXt -lX11 -llo -L/usr/X11R6/lib -L /usr/local/lib/lo
```

Now we are ready to see if we can get an OSC packet from the game. A useful

utility is a little program called `dumpOSC` that will display any received OSC messages to the console. You can get it from here:
http://www.cnmat.berkeley.edu/OpenSoundControl/dumpOSC.html

Run `dumpOSC` in the background and then start the Xinvaders game. You should see a message sent from game on startup. Alternatively use netcat with verbose, UDP and listen flags, `netcat -ul localhost -p 7770 &` - but be mindful that the last float in our message will not display correctly as an ascii char.

```
dumpOSC 7770 &


dumpOSC version 0.2 (6/18/97 Matt Wright). Unix/UDP Port 7770
Copyright (c) 1992,96,97,98,99,2000,01,02,03 Regents of the University of
California.


$ ./xinvaders
/foo/bar "create" "invaders" 1.000000
```

Adding OSC sound hooks

Now we know the game works and we have OSC connectivity, let's begin adding the sound hooks. To save some time looking through all the source code lets search it for likely functions. What we are looking for is conditions where something is "hit", "explodes" or other "events". Eventually we are going to have to read through all the files but let's begin with a quick search on a likely term.

```
grep -in destroy *.c


base.c:25:Boolean basedestroyed;
base.c:100: basedestroyed = TRUE;
base.c:141:void DestroyBase()
base.c:145: basedestroyed = TRUE;
base.c:158: if(!basedestroyed && BaseNearPoint(base, x, y)) {
base.c:159: DestroyBase();
base.c:187: basedestroyed = FALSE;
base.c:202: if (basedestroyed) {
base.c:218: basedestroyed = FALSE;
shot.c:69:static void DestroyShot(i)
shot.c:78:static void DestroyVshot(i)
shot.c:108: DestroyShot(i);
shot.c:132: DestroyShot(i);
shot.c:163: DestroyVshot(i);
spacers.c:70:static void DestroySpacer()
spacers.c:91: DestroySpacer();
vaders.c:106:static void DestroyVader(vader)
vaders.c:141: DestroyVader(vader);
widget.c:288:static void Destroy() {}
```

Well, that is very promising. Right away we find a number of interesting functions that will probably lead us to points in the code where we can send an OSC message to create a sound. I will list these in a moment but since they are in other files than `main.c` you need to add the following line to each of the other source files so that our OSC port handle is visible.

In the other source files declare the handle

```
30 extern lo_address t; /* OSC port address for sounds */
```

Now we will find one of the events for which we wish to add a sound. In the sourcefile `vaders.c` we discover a function called when an enemy is destroyed. That seems like a good place to add an explosion effect. Here is the complete function listing. Notice our new line of code is added at the bottom, just after the explosion graphics are painted. The message destination is `/game/invaders` and "sf", "invader-destroyed" is just a label meaning sound-function for the exploding enemy ship. We will define the meanings of these fields later when we use them, for example we might use the float part of the message to be the sound effect volume or intensity.

Exploding enemy

```
static void DestroyVader(vader)
Vader vader;
{
PaintVader(vader, backgc);
score = vader->value;+
PaintScore();
numvaders--;
vaderwait /= 2;
vader->alive = FALSE;
vader->exploded = TRUE;
PaintExplodedVader(vader, vader->gc);
lo_send(t, "/game/invaders", "sf", "invader-destroyed", 1.0f);
}
```

Now, every time a enemy is destroyed we will send a message through OSC to our sound development platform. You may use Reaktor™ from Native Instruments or Max/MSP™ from Cycling74 or Puredata. We will use Pd because it is the most flexible of the three environments. Before we move on to designing the sounds add messages at or near these points in the source files. The call to send the sound message will usually be the last line in the relevant function.

Event bindings

Line number	In file	Sound effect
148	base.c	base-destroyed
295	base.c	building-hit
125	vaders.c	invader-destroyed
133	shot.c	shot-hit-shot
181	shot.c	add-shot
197	shot.c	add-vader-shot

Creating sound effects

Start Puredata on the same machine as you are running the game, we will use `localhost` as the destination for all these experiments. Begin a fresh patch and add a `dumpOSC 7770` object. This will pick up the OSC messages for us. If you get a problem creating this object check that the port 7770 isn't still in use with `lsof | grep 7770`. Did you remember to kill the dumpOSC command? Now we will connect the `dumpOSC 7770` object to a `route /game/invaders` in order to filter out the messages for each sound effect. If a message matches it will appear at the lefthand outlet, remaining messages will flow through to the rightmost outlet. So, by chaining together a cascade we can route all the messages to individual sub-patches like this:

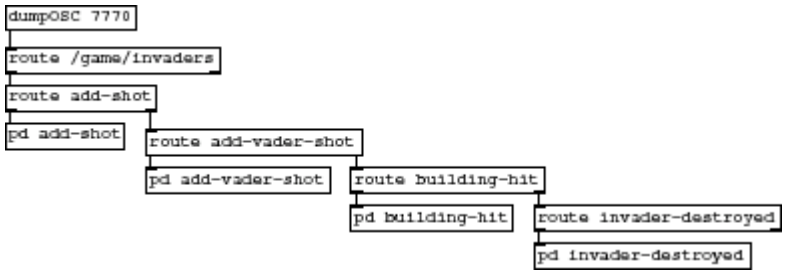

Figure 2: Routing messages

All we have to do now is work on the sound effects. Start the game running and add a subpatch for the weapon sound as below.

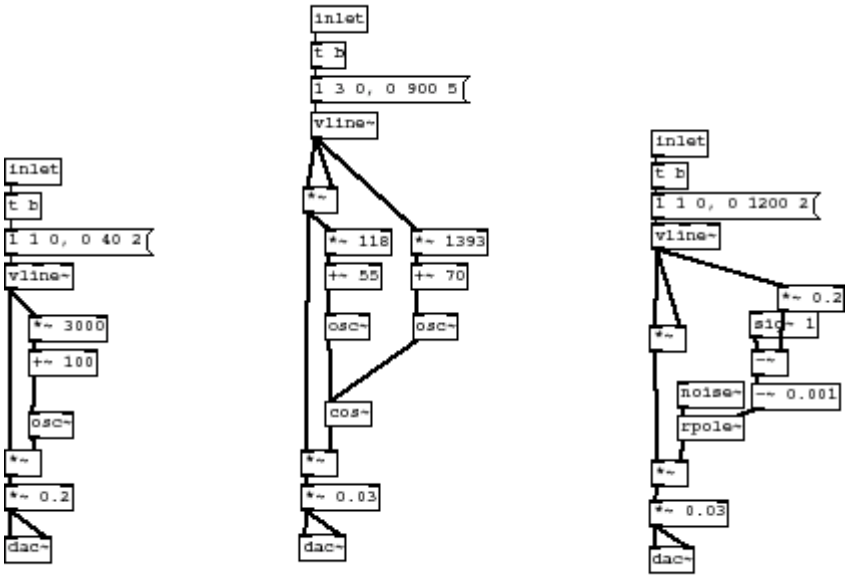

Figure 3: Sound effects for three events

Note how the flexibility of this approach is now showing through, we can pause the game and work on sound design then return immediately to the game without recompiling anything, or having to leave and return to any environment. In fact we can work on the sound design while the game is playing. Further suggestions are given below for the enemy laser sound and the exploding building. Note that these are deliberately designed to sound a bit crummy and "8-bit", realistic sounds are not what we desire for a 70s arcade game. We will not delve into the synthesis details too deeply here, plenty of information to understand this is available elsewhere on http://obiwannabe.co.uk/. Enemy shots use use FM to give a richer wobbly effect. For building and enemy explosions a burst of swept noise is used. Tune the filter to sweep over the low ranges between 2000Hz and 100Hz and add plenty of clipping distortion for an old style arcade game explosion.

About this document ...

Hooking game audio: Using Pd and OSC for rapid DSP development

This document was generated using the **LaTeX**2HTML translator Version 2002-2-1 (1.70)

The command line arguments were:

latex2html `-split 0 -local_icons hooking.tex`

The translation was initiated by root on 2007-10-11

Next Group Up Previous

Andy Farnell
http://obiwannabe.co.uk