TabSynth:
A graphical tablet based synthesizer

Cosmin Deaconu

Music 420

March 11, 2009

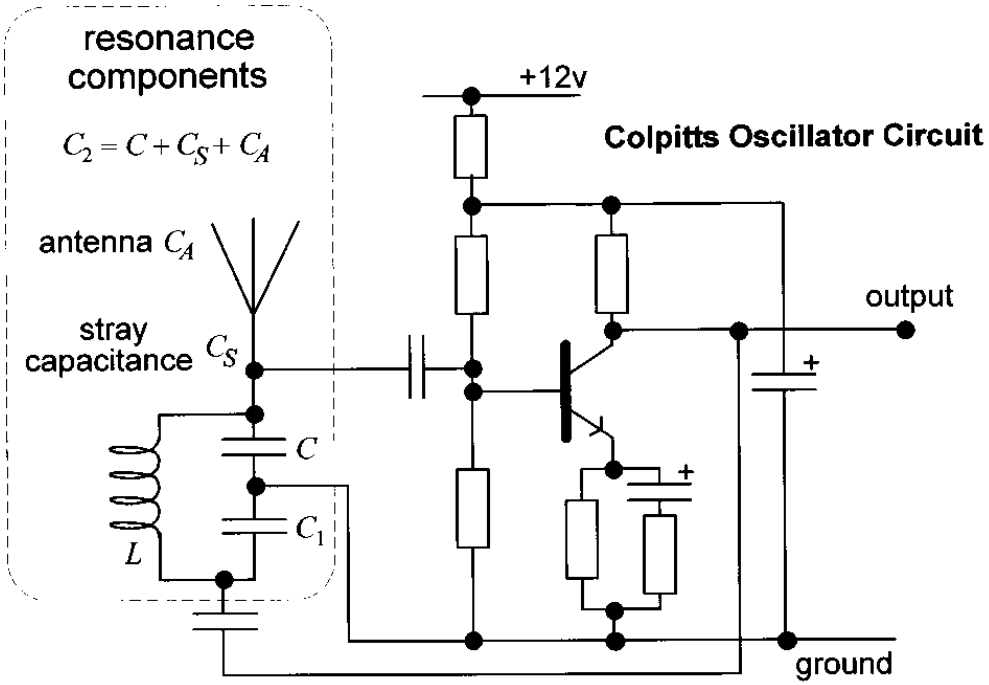
Introducing TabSynth 0.1

- Uses position and pressure data from a graphical tablet to make (pretty?) sounds.
- Written in C++ using the STK for sound and gtkmm for the GUI / input.
- Known to compile on Ubuntu and Fedora. In theory, should work on any POSIX system (with perhaps some changes to the Makefile). .
- Dynamically loads synthesis plugins at runtime (using dlopen). Not very many plugins right now, but it is easy to write one!
- Code available in mercurial repository. Check out with the command
  `hg clone http://freehg.org/u/cozzyd/tabsynth/`

Obvious application: Modeling the Theremin

- Included in the distribution is an attempt at a physical model of a theremin
- Based on *Physics of the Theremin*

  Kenneth D. Skeldon, Lindsay M. Reid, Viviene McInally, Brendan Dougan, and Craig Fulton, Am. J. Phys. 66, 945 (1998), DOI:10.1119/1.19004
- We hear the beat frequencies between two really high frequency oscillators.



Calculating the pitch

Used the following equations for one oscillator, where L is inductance, C_1, C_2 are specified capacitances, h is height, d is diameter, x is distance from hand, k is a factor having to do with the distance from the ground ($\sim$0.4 for slightly above ground) and K is a fudge factor having to do with the human hand not being an infinite plane.

$$f = \frac{1}{2\pi\sqrt{L}}\left(\frac{1}{C_1} + \frac{1}{(C_2 + C_A)}\right)^{(1/2)} \tag{1}$$

$$C_A \approx \frac{2\pi\epsilon_0 h}{\log(2h/d) - k} + \frac{\pi\epsilon_0 h}{K\log(4x/d)} \tag{2}$$

k was set to 0.4 and d to 0.01 m. Everything else is modifiable! The volume is varied by the pressure of the pen.
The other oscillator is tuned such that the difference is 0 hz at the edge of the screen.

The Plugin System

- At runtime, TabSynth will load all .so files in the plugins directory.
- A plugin completely defines the output sound.
- A plugin has access to the tablet parameters (position, pressure, etc.) as well as any number of options that can be set by the GUI. Whenever a GTK event occurs in the input window, the plugin is informed of the changed parameters.
- The audio generation runs in a separate thread, which calls tick() on the plugin.
- The plugin can also draw an arbitrary figure in the input window (using cairo).
- In the next few slides, I will guide you through a really simple plugin.

Really simple plugin

```
//super_simple.cpp

//A really dumb plugin for demonstration purposes

#include "../tabsynth_plugin.h" //This file must be included
#include <stk/BlitSaw.h>        //Use this for synthesis
#include <cmath>

class SuperSimple : public TSPlugin {

  protected: //Protected Member Variables
    double vol;
    BlitSaw * blit;
```

Really simple plugin: Constructor/Destructor

```
public:
  SuperSimple() { //Constructor
    blit = new BlitSaw(220);
    vol = 0;

    //Add an option to set the number of harmonics
    options.push_back(new PluginOption(OPTION_SPIN,
           "NumHarmonics", "setHarmonics is called
            with this number", 10,1,100,50));
  }

  ~SuperSimple() { //Destructor
    delete blit;
  }
```

Really simple plugin: setParameters()

```
//Tablet event occurred. Calculate new frequency.
//Return frequency for display purposes
 StkFloat setParameters(double x, double y, double pressure,
                        bool button2, bool button3,
                        double tiltX, double tiltY) {
//Some weird way of computing frequency
double freq = 1000*pow((x*x-y*y+x*y-pressure*pressure),2) ;

//Use pressure as volume
vol = pressure;
blit->setFrequency(freq);

//Grab the harmonics from the GUI option
blit->setHarmonics( (unsigned int) options[0]->getValue());
return freq;
}
```

Really simple plugin: tick() and draw()

```
//Computes next sample
StkFloat tick()  {
   return blit->tick() * vol;
}

/* We could draw something here, but Cairo is complicated,
   so we'll skip this*/
void draw(Cairo::RefPtr<Cairo::Context> cr, int size) {}
```

Really simple plugin: metadata methods

```
  std::string getName() {
       return "Super Simple";
  }

  std::string getDescription() {
       return "A toy plugin for demonstration";
  }

  std::string getVersion() {
       return "-1";
  }

}; //End of Class Definition
```

Really simple plugin: getting it to dynamically load

```
/* In addition to containing a class inheriting from TSPlugin,
 * a plugin must have the following two methods. (Used for
 * dynamic loading) */

extern "C" TSPlugin * init() {
     return new SuperSimple;
}

extern "C" void destroy(TSPlugin * p) {
     delete p;
}
```

Really simple plugin: that's it! (sort of)

- My build system is very rough around the edges now. Have to put simple_plugin.cpp in src/plugins and then modify the Makefile in src/plugins (I haven't yet figured out how to make those automagic Makefiles that automatically process everything).
- Making will copy the .so file into the plugins/, which is where tabsynth looks for it.
- In principle, you could distribute the .so file. However, it will only work on platforms similar to yours (and until I decide to change the ABI =)).
- If you need to use NoteOn / NoteOff, the easiest way is to notice when pressure changes from 0 to non-zero and vice versa (see Flutemin.cpp).

Known Issues

- No event is triggered when the cursor leaves the drawing area. Does anybody know how I'm supposed to detect this using GTK?
- Glitchy audio when volume changes too rapidly. Need to fade out volume probably.
- Memory leaks
- Subpar build system (with probably many unneeded flags)
- Crash on exit (double free or something in Glib?)
- Poorly structured, poorly documented, hacked-together code.
- Resource / plugin paths hard linked. The executable has to be executed from the right place to work. Does anybody know the proper way to fix this?
- While gtkmm and stk are cross-platform, I use POSIX-specific paths and the POSIX dynamic loading functions (dlfcn.h). Also I link to -lasound.
- Not enough plugins :((I spent too much time getting the infrastructure working, so I didn't have time to do the fun part).