virtualdub.org

Proof that I had too much free time in college

Current version

v1.10.4 (stable)

Navigation

Main page
Archived news
Downloads
Documentation
- Capture
- Compiling
- Processing
- Crashes

Features
Filters
Plugin SDK
Knowledge base
Contact info
Forum

Other projects
- Altirra

Pitch shifting §¶

I have a lot of miscellaneous junk lying around my hard drive's `projwin` directory, which is where I keep most of my VC++ projects. A quick glance:

- a fanfiction reader with built-in phrase search across all files and suspend/resume
- a Final Fantasy VII model viewer
- an infinite-playfield version of Conway's Game of Life
- an NTVDM driver to emulate an SBPro in a Windows 2000 dos box

All very unfinished, of course.

One pair of projects that I've hacked on sporadically is an MP3 player with a spectrum analyzer and a WinAmp plugin, which share a simple audio processing library in common. I've experimented with a few audio effects with these; most of them were failures, particularly all of my attempts at reverb. I did have some mild success with some frequency-based algorithms, namely the "center cut" filter that is now in VirtualDub.

A filter that I've been trying to get working well for some time is a pitch shifter. VirtualDub has an experimental one, the "ratty pitch shifter," but as you might guess it's not very good. I recently found an algorithm that works better, though.

What's a pitch shifter?

A pitch shifter is a device or algorithm that stretches or compresses the frequency spectrum of an audio signal. The result is that all sounds in the audio are moved together to higher or lower pitches. This is done by scaling all of the pitch frequencies so that harmonic relationships are preserved — that is, the 3rd harmonic of a sound is still at 3x frequency after the scaling operation. (I suspect that what I'm describing is actually a pitch scaler and that a pitch shifter actually just slides all sounds up and down in frequency, destroying harmonics, but I hear the term pitch shifter used too often.) The result is that you can take a soundtrack and make the whole track

Search

go!

Archives

01 Dec - 31 Dec 2013
01 Oct - 31 Oct 2013
01 Aug - 31 Aug 2013
01 May - 31 May 2013
01 Mar - 31 Mar 2013
01 Feb - 29 Feb 2013
01 Dec - 31 Dec 2012
01 Nov - 30 Nov 2012
01 Oct - 31 Oct 2012
01 Sep - 30 Sep 2012
01 Aug - 31 Aug 2012
01 June - 30 June 2012
01 May - 31 May 2012
01 Apr - 30 Apr 2012
01 Dec - 31 Dec 2011
01 Nov - 30 Nov 2011
01 Oct - 31 Oct 2011
01 Sep - 30 Sep 2011
01 Aug - 31 Aug 2011
01 Jul - 31 Jul 2011
01 June - 30 June 2011
01 May - 31 May 2011
01 Apr - 30 Apr 2011
01 Mar - 31 Mar 2011
01 Feb - 29 Feb 2011
01 Jan - 31 Jan 2011
01 Dec - 31 Dec 2010
01 Nov - 30 Nov 2010
01 Oct - 31 Oct 2010
01 Sep - 30 Sep 2010
01 Aug - 31 Aug 2010
01 Jul - 31 Jul 2010
01 June - 30 June 2010
01 May - 31 May 2010
01 Apr - 30 Apr 2010
01 Mar - 31 Mar 2010
01 Feb - 29 Feb 2010
01 Jan - 31 Jan 2010
01 Dec - 31 Dec 2009
01 Nov - 30 Nov 2009
01 Oct - 31 Oct 2009
01 Sep - 30 Sep 2009
01 Aug - 31 Aug 2009
01 Jul - 31 Jul 2009
01 June - 30 June 2009
01 May - 31 May 2009

down lower or higher in pitch without changing its speed, which is what happens if you simply resample it (like speeding up or slowing down a tape).

What's neat about having a workable pitch shifter is that if you combine a pitch shifter with a resampler so that the pitch changes from the two cancel out, you affect speed instead. In VirtualDub, you can do this by combining the "ratty pitch shift" with the "stretch" filter, and give both the same factors. You can then speed up or slow down a video by changing the video frame rate to match. One use for this is correcting for the ~4% speed error when converting between 23.976 fps (FILM) and 25 fps (PAL).

Also, a lot of music sounds neat when you shift it down a few semitones, although any farther than that and you get serious chipmunkiness due to distorting the voice spectrum.

The ratty pitch shifter

The RPS is a time domain pitch shifter — it works by slicing up the audio track into sections and then overlapping them to produce a slightly shorter or longer track. That produces a speed change, which is then turned into a pitch change through a post-resampler. Because it works on relatively long sections of audio, around 40ms, it doesn't affect the pitch of sounds during the first pass. The problem is that the splices are often less than ideal, as you often hear hiccups or volume distortions at the join points. When pitch shifting up you often hear sounds being repeated, and when shifting down you can hear sounds occasionally omitted.

Another problem with time-domain shifters is that they shift sounds back and forth in time slightly. This isn't a big deal for an individual sound; it is a problem when you are working with stereo audio and a sound that's supposed to be in the center ends up at two different positions in the stereo channels. Sounds that are supposed to be in the center suddenly turn into echoes, since they seem to be bouncing around you. I found an easy trick to prevent this from happening: instead of shifting left/right, shift sum and difference instead. Sounds that are supposed to be in the center stay there, and existing echoes/reverbs are mostly preserved. The "ratty pitch shift" does this automatically; for other algorithms, the "butterfly" audio filter will do the necessary sum/difference

computation.

Despite my awful attempts, time-domain pitch shifting can work well. The pitch shifter filter in the SoundBlaster Live! is an example. The EM10K DSP in the SBLive resamples all voices to 48KHz and then pushes them through the main effects engine, which executes 512 instructions per sample (1.77MHz). It has a single-cycle multiply-accumulate instruction, so I suppose in theory it could do a small Fourier transform, but somehow I doubt that's what the pitch shifter does as it converts everything to mono and imparts a bit of an echo. Summing overlapping windows is a lot cheaper; it's just a question of how exactly you place and window the taps, which is something I haven't figured out yet.

Frequency-domain pitch shifting

The alternative is pitch shifting in the frequency domain. This means working in the frequency spectrum; think of the bars in a spectrum analyzer, except a lot finer and having phase as well as amplitude per frequency. Frequency domain pitch shifting isn't that hard in its basic form, and I had some success pitch-shifting up by about 5% just by point-sampling the spectrum, but attempting to up more or down at all just didn't work. It also takes a decent amount of computing power to do the transforms; this was a bit of a problem back when I had a Pentium 120. Nowadays a Pentium 4 or Athlon 64 has so much floating-point power you can code a crappy, unoptimized FFT-based audio algorithm and still test it in real-time.

One of the problems is that when you do a Fast Fourier Transform (FFT) to convert from time to frequency domain or back, you still have to pick the size of the transform, and that determines a window of time that you're working with. For instance, a 2048-point FFT on 44KHz audio covers about 50ms of audio. That unfortunately means that the frequency domain data you get out also has a bit of position encoded into it within that window. The result is that when you play around with the magnitudes and phases in the frequency spectrum, you can end up "smearing" sharp sounds by spreading out its frequency components. An example would be a cymbal clash that turned into a mushy pop. Frequency-based compression algorithms like MPEG layer III suffer from problems like this as well;

handling such "transients" well is a tricky problem.

Another problem has to do with how the FFT is used to process audio. Processing an entire track in one big transform is computationally prohibitive, as the FFT's computational complexity is O(N log N), and also disadvantageous since you then receive a ton of positional information in the spectrum. What is done instead is to split the signal into a bunch of small sections that overlap by some factor, such as 2x or 4x, and then smoothly scale them using a window so that the sections blend into each other nicely. Unfortunately, messing around with the spectrum can give you a result that sounds decent in a single section, but then interferes with adjacent sections when you sew them together, resulting in an audible beat in the output. This happens when the phases of the sine waves don't match. I discovered this when I attempted to linearly interpolate when resampling the spectrum, which didn't work well at all.

An algorithm that seems to work well

The DSP Dimension has a frequency-domain algorithm for pitch shifting with included source code that sounds pretty good. Basically it watches phase over time at each frequency and uses that to determine how much it should skew the phases on the sine waves on the output. As usual, the source code is not well optimized as it is meant for tutorial purposes. There also seem to be some small miscellaneous problems in it, such as a root-two gain factor in the output and some slight bin index errors in the way the spectrum is point sampled. Nevertheless, the algorithm sounds good, and is one that I hope to incorporate into VirtualDub.

With due credit, of course.

Speed is not so good at the moment. I fished out the obvious unnecessary divides and got it down to about 50% CPU on a 1.6GHz Pentium 4; I hope to optimize the FFT some more and/or convert it to a Fast Hartley Transform (FHT) to bring that down. I did the classic factoring optimizations for the first few stages, which have lots of zero, one, and root-two factors, until I realized that I still hadn't done the obvious optimization of converting the complex FFT to a real FFT. FFTs used to really scare me, but after doing a bunch of IDCT hacking for JPEG/MPEG work as well as a few iterations of FHTs and some 3D graphics, I've

gotten more comfortable with them. You eventually get to the point where you see something like this:

```
x2 = x*c - y*s;
y2 = x*s + y*c;
```

...and immediately see a rotation.

I know there are lots of good FFT packages out there, with FFTW coming to mind, but I'd rather hack on this instead of integrating in stuff. In general, I don't like throwing together packages to make programs; you still end up spending time doing integration work, and in exchange for possible time saved coding you get build process and distribution headaches. It's also not as fun or educational.

An additional issue is that because the algorithm needs actual phase angles, it requires a square root and arctangent on input and sine/cosine on output. These are about 30, 90, and 100 clocks respectively and are all expensive operations, i.e. bottlenecks. Sometimes there are ways to use vectors and rotation matrices to bypass these. I don't see an easy way to do so in this case, though, and I'm not sure how fast I could do an atan2() approximation with a polynomial expansion or similar technique.

There's also the opportunity for SSE/SSE2 optimization, of course, but that's for later. At some point I might try an MMX FFT as well; precision would be marginal even with `pmaddwd`, but unlike fast IDCT algorithms, all of the coefficients are less than 1, which is less worrisome.

10 comments | Apr 07, 2005 at 00:30 | default

Comments

Comments posted:

---

so you're writing your own NTVDM sound driver? why not use an existing one like http://sourceforge.net/projects/vdmsound

apexo - 07 04 05 - 03:39

The pitch-shifting filter that comes with the opensource "Audacity" seems to be quite good. [http://audacity.sourceforge.net/] It might be worth a look.

Cranston Snord - 07 04 05 - 06:30

As you might have noticed I used SoundTouch for the TimeStretch function in AviSynth that produces quite good quality - it might also supply you with inspiration, now that you doing this from scratch.

http://www.iki.fi/oparviai/soundtouch

Klaus Post (link) - 07 04 05 - 07:07

---

apexo:
I didn't know about VDMSound when I started that project; I'm not working on it anymore. One big problem is that the NT dosbox is terrible and doesn't emulate nearly as much as it should. For example, attempting to set timer 0 to 18.2Hz to 60Hz, a common operation, causes your DOS program to receive interrupts at 60Hz on average, but in bursts every 18.2Hz. Unfortunately, you can't fix these problems from a VDM plugin. It was still neat to see the DSP commands being sent to the audio device, though.

Phaeron - 07 04 05 - 23:43

Hi,
you might know it already but concerning FFT there is a very nice open-source library out there called FFTW. It's incredibly fast (at least for larger data-sets, don't know about 2048 points) and relativly easy to use. Check http://www.fftw.org. They have done the SSE1/2 optimizations and they also implemented a large varitey of different algorithms to choose from.

bye...
...manilius

manilius - 08 04 05 - 11:04

Ugh, sorry for not even reading your full post, you mention FFTW yourself. However, I can recommend it, even if just to look at the source code.

bye...
...manilius

manilius - 08 04 05 - 11:07

Hi,

funny, some time ago I was looking very hard for a FF

VII model viewer on the internet, and the only one I found didn't work right. I did some reverse engineering myself but didn't get very far. How about putting up a sf.net page with the (unfinished) source code? ;-)

Darkstar - 08 04 05 - 14:09

I like djbfft, at http://cr.yp.to/djbfft.html. Easier to integrate than FFTW, IMO, and easier to license (i.e. can use for non-gpl compatible software)

Justin Frankel (link) - 16 04 05 - 15:32

While working with the sound, (I'm a sound engineer with C/C++ programming knowledge), I've been trying many algorithms on time-stretching/pitch-shifting.

And for realtime I came to prefer PSOLA. You might try to check it out:

http://en.wikipedia.org/wiki/PSOLA

And sorry about replying about several years late, you might have discovered this one yourself.

HMage (link) - 12 06 06 - 11:38

I too am replying several years late, but I thought you might find the Pacemaker Winamp plugin interesting...

http://www.surina.net/pacemaker/

Using this plugin, you can modify the tempo, pitch, and/or speed of a song in real-time while it's playing.

I've played with it a lot and created some interesting sounding versions of popular songs.

My only gripe is that the quality can sometimes be very bad.

Krick - 12 05 08 - 13:36

Comment form

Please keep comments on-topic for this entry. If you have unrelated comments about VirtualDub, the forum is a better place to post them.

Name:

Email (Optional):

Your email address is only revealed to the blog owner and is not shown to the public.

URL (Optional):

Remember personal info?
- Yes
- No

Comment:

An authentication dialog may appear when you click Post Comment. Simply type in "post" as the user and "now" as the password. I have had to do this to stop automated comment spam.

Post Comment | Preview Comment

Small print: All html tags except <b> and <i> will be removed from your comment. You can make links by just typing the url or mail-address.