

TimeWarp

by Robert Poor, NeXT Computer, Inc
September 1992

Overview

As powerful as it is, the NeXT Sound Kit™ doesn't give you many options for manipulating digital audio sound in real time. On the other hand, programming at the Sound Driver level is complicated.

To simplify matters, we've created a **DACPlayer** object that makes it easy to play digital audio sound through the built-in DACs on the NeXT computer. Using the DACPlayer, you get a chance to manipulate sound data on the fly as it is sent to the DACs.

TimeWarp is a simple application that demonstrates one way you can use the DACPlayer object. TimeWarp plays a sound file with a "speed control," much as you might find on a tape recorder, that lets you speed up and slow down the sound as it is played. This version of TimeWarp uses simple linear interpolation between samples to change the effective playback rate. While this approach has theoretical problems (interpolation is a form of low-pass filtering, you're subject to aliasing artifacts, etc), it sounds just fine for most sound sources and it's computationally simple.

We use fixed point arithmetic in the inner loop, which interprets an int as a value with 1 sign bit, 15 bits of "whole part" and 16 bits of fraction.

About DACPlayer:

The file DACPlayer.h is well commented, but here are some additional notes on the use of the DACPlayer object:

You will always set a delegate for the DACPlayer object, and at the very least, you will implement a playData::: method. When the DACPlayer is run, it will call the following delegate methods:

- willPlay :player;

Called by the player when going from a stopped to a paused state. Called after all the resources have been allocated but

before any regions get queued. The various dacPlayer configuration parameters MAY be set from a willPlay: method.

- didPlay :player;

Called when the player goes into a stopped state (either from a stop or abort message) after all the sound resources have been freed.

- playData :player :(char *)data :(int)nbytes;

Called whenever the player wants more sound data. player is the dacPlayer requesting the data, data is a buffer of length nbytes. The buffer is guaranteed to be zero'd out. The dacPlayer requires that the samples you write are stereo 16 bit samples. The samples will be played at whatever sampling rate you established in a call to setSamplingRate:

- didChangeState :player from:(Pla_state_t)oldState to:(Pla_state_t)newState;

Called whenever the player changes state. This method can be used to update status displays or perform certain operations, for example, when the DACPlayer stops.

Programming tips for DACPlayer:

You never need to call the prepare method directly ± you can always call the pause method instead. If the DACPlayer is in a stopped state, calling pause will call prepare for you.

A good time to configure the DACPlayer parameters (regionSize, regionCount, samplingRate) is from within the willPlay: delegate method.

Call finish rather than stop if you want any queued data to get played rather than stopping immediately.

The DACPlayer can only play what the built-in DAC can play, i.e. 16 bit stereo samples at sampling rates of either 44100 KHz or 22050 KHz. (I don't mention that anywhere else, do I?)

In theory (I haven't tested this), you can monitor how well the system is keeping up by watching how full the buffer queue stays. In a call to `playData:::`, compute `headroom = [player samplesQueued] - [player samplesPlayed]`. If headroom ever hits 0, you know that your `playData:::` method has fallen behind in serving up samples to the sound driver. (Did you hear a click?)

Changes since the last DACPlayer:

The previous version of DACPlayer didn't distinguish (very well) between data that had been queued with the sound driver and sound data that has been played. In particular, there was no clean way to stop the DACPlayer without cutting off the last few buffers of sound data. Where before there was a single `stop:` method, there are now two methods:

- `stop`;

Calling the `stop` method will stop the DACPlayer immediately, and any data that has been queued up is discarded.

- `finish`;

Calling `finish` will shut down the DACPlayer gracefully. If the DACPlayer state is `PLA_RUNNING`, then the DACPlayer will stop enqueueing new regions (and will stop calling `playData:::`) and will set the state to `PLA_STOPPING`. Only when the last available buffer has been played by the sound driver will the DACPlayer stop. A typical place to call the `finish` method is from within the delegate's `playData:::` method when all incoming data has been processed.

There are several new methods that make it easier to find out how much data is queued and how much data has actually been played since the last call to `prepare`:

- `(int)bytesQueued`;
- `(int)samplesQueued`;
- `(int)framesQueued`;
- `(double)secondsQueued`;
- `(int)bytesPlayed`;
- `(int)samplesPlayed`;

- (int)framesPlayed;
- (double)secondsPlayed;

The priority of the DPSSAddPort mechanism has been raised from NX_BASETHRESHOLD to NX_MODALRESPTHRESHOLD
- this means that you can grab a slider with the mouse and the sound won't stop.

Future Directions:

A motivated programmer and lover of sound will consider the following rainy day programming tasks:

Find and squash any bugs. (Did I say it was perfect?)

Make the necessary modifications so the application will compile cleanly under NeXTSTEP Release 3.0.

Build in support for international language support and localization.

Create a "Player" superclass and implement a "FilePlayer" subclass in addition to the DACPlayer. A FilePlayer would have the same interface as the DACPlayer (except for some different configuration-setting methods) and would behave like the DACPlayer, but it would write its data to a file rather than the the DACs.

Create other subclasses of Player that will send data out the DSP port to devices such as:

- Singular Solutions A/D64x (AES/EBU and S/PDIF outputs)

- Stealth Technologies ADA1800 (AES/EBU and 16bit DAC outputs)

- Ariel ProPort 656 (16bit DAC outputs)

- Ariel DATPort (AES/EBU and S/PDIF outputs)

Search out and quash any "big-end/little-end" dependencies - NeXTSTEP 486 is coming.

Create a cool icon for the App.

Other References

Valid for 2.0

Not valid for 3.0 (compilesokay, but should use updated headers and Project Builder)