SC Ugens 2

Buffers, Triggers, Demand and Delay lines

Assignment 4

Assignment 3

Implement a simple ***system*** that allows its user to create simple compositions using SuperCollider. The system needs to contain the usage of ***two custom UGens*** written in C++. One of the UGens should be used for ***control information*** and the other one for ***synthesis*** or ***sample playback***.

The system should offer different ways to combine the two in order to create sound material that could be used in a composition.

The implementation should reflect a *sensitivity* of how to design compositional systems with SuperCollider.

The assignment should be handed in before the the ***20st of June***. (and will be discussed during this class).

UGens

SimpleTrigger

In SuperCollider many unit generators act upon ***triggers***. A trigger is usually a transition from *zero to a higher value*. This means that the behavior of the recieving UGen changes once it receives a number > 0 when a previous value was <= 0.

SimpleTrigger demonstrates the principle of a receiving trigger UGen.

Its function is very simple for reasons of clarity. All SimpleTrigger does is to output a *new random number* on each trigger, otherwise it returns the previously generated random number.

So if a transition from less or equal than zero occurs, a new random number is created and returned until another such transition occurs.

SimpleLooper

SuperCollider uses shared ***buffers*** for things such as sound samples. These buffers can then be *accessed* and used by different UGens to create sounds.

SimpleLooper demonstrates how one can read an allocated buffer and play it back. Additionally it will loop the playback. It always starts from the beginning of the sample but contains an argument for the end of the loop. If the end is not specified it will loop the whole buffer.

Important properties about the buffer are obtained using the *GET_BUF_SHARED* macro. This macro will use the ***unit struct*** to fill it with the required info. Therefore a specific naming convention needs to be followed in order for it to know where to put this information.

SimpleLooper

The struct needs a field called ***m_fbufnum*** and a field called ***m_buf***.

The macro declares variable ***fbufnum*** and assigns it the value read from the first argument.

This value will default to 0. The macro declares a pointer to the buffer data (***bufData***) and also declares ***bufChannels***, ***bufSamples***, and ***bufFrames***.

SimpleDemand

Demand rate UGens operate differently from regular-rate UGens and only return values when *demanded*. In practice this means that they do not fill output buffers with values but only return a single value.

SimpleDemand demonstrates how demand rate UGens works by simply *returning a random number* in a range set by the user with hi and low values.

The numSamples is not used. Demand rate UGens also need to inherit from DUGen.

SimpleEcho

Being able to allocate ***dynamic memory*** is essential to the creation of many UGens. Delays are an example where dynamic memory is needed.

SimpleEcho demonstrates allocating memory with a ***delay line***. The process will simply write incoming data to a delay line and read it slightly faster at a different position.

Additionally it will *mix* the incoming sound with the delayed sound.

The UGen also uses a ***deconstructor*** that will take care of freeing the memory allocated for the delay line. Calling the deconstructor is handled by SuperCollider.

UGens using a destructor need to register with DefineDtorUnit.

Exercises

Exercises (SC UGens)

1. Extend SimpleTrigger by letting it output random number set by a range that is defined by a maximum and minimum value. The range is what should change at each trigger and cloud itself by set from randomness or by changing a previous range.

2. Extend SimpleLooper so that it will accept as argument the start position of the buffer to be looped. Additionally each loop iteration should have a slight attack/decay envelope in order to prevent clicks.

3. Extend the demand rate UGen so that it outputs a ramp that goes from -1.0 to 1.0 and can be audible as a waveform.

4. Extend the delay line by moving the reading position

Previous Exercises

Exercises (SC UGens)

1. Implement a Sample-and-hold UGen that generates a random numbers that are held for a specific amount of time.

2. Implement a random generator UGen that produces random numbers distributed within a set range. These ranges could be modified so that there is a high range and a low range. The output should then oscillate between the two.

3. Write a UGen that generates pulse waves similar to the program that created a sawtooth waveform presented. The pulse width and period should be set by variables that can be modified before the program runs.

4. Write a UGen for brownian motion that takes a random step from an initial position at each calculated sample.

Exercises (Port Audio)

1. Extend the sawtooth program by making the frequency on the right channel higher (for example double) compared to the left channel.

2. A simple way of downsampling a signal is to repeat previously calculated values. Extend the white noise program by repeating previously generated random numbers at least once (or more often).

3. Write a program that generates pulse waves similar to the program that created a sawtooth waveform presented. The pulse width and period should be set by variables that can be modified before the program runs.

…

Exercises (Port Audio)

4. Extend any of the three examples by adding an amplitude envelope. The envelope could be added to the data structure as an array of floats that audio is then multiplied with. The envelope should have at least an attach, a sustain and a release period.

5. Extend the buffer record and playback example by making it play two layers of sampled sound where the second one has a different starting position and potentially a different playback duration.