Technical Report on the design of 5 sonic toys created using Pure Data

Andrew Reeman

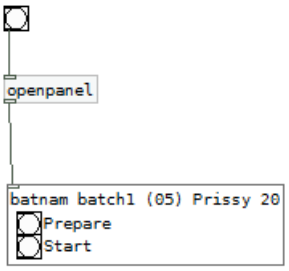
October 31, 2011

Contents

1 Introduction

The majority of these presented patches share a key design concept which is the automatic generation of abstractions. This was used in 'Birds' to generate the bird abstractions; 'Compos' to generate the composite parts and 'Pulse' to generate the customisable pulses. While 'batnam' does not host this concept it is designed in such a way that it is possible to have multiple instances. In this respect the design of batnam held a similar idea.

2 Batnam



Summary

Like Birds and Compos this patch was designed to fulfil a personal creative need: to be able to process multiple files in a directory via pure data while taking advantage of the naming scheme of 'batch-formats' [1]. Batnam is a read/write batch processor for files already in the batch-format. It is possible to have multiple batch file 'families' [2] in one directory. Batnam will restrictively read only one family within the batch limits set by the user. It is a multi-instance object so it is possible to run two simultaneously. If the user creates the relevant 'catch~' objects then batnam will write this input to new files (named with respect to the files read).

Design Features

The user must input a string that directs to the starting file. The first argument is the unique name of the batnam object. The second is the batch-format and also the maximum number to be read. This must be placed between parentheses. The third argument, depending upon the operating system, is either the new directory or prepended name for the written files. The fourth argument is the delay (in ms) between each file that is read. The main function in this patch is that it will split up the input string into three components:

- Pre-batch number: This will usually include the file path and the generic name. ***/usr/home/harry****004.wav*
- Batch-number. */usr/home/harry****004****.wav*
- Post-batch number: This will usually be the file format but could also include the generic name. */usr/home/har004****ry.wav***

[1] I define a batch format as a file that follows the following format: (generic name)-(ascending numbers using a consistent amount of digits (e.g. 001, 002, 003. Or 01, 02, 03)-(file format)

[2] A family is simply a group of files that share a generic name with varying batch numbers

These components will be sent to a 'readsf~' object with only the batch-number being modified. The same method is used for writing a new file. The locations of the components are found by splitting the string into individual characters then sequentially searching the string for floats and symbols. The split points are found using the information in the second argument (this is the digit count to search for). With the string successfully split the next phase is to modify the batch-number. This is done commencing from the batch-number in the input string up until the number in the second argument. This modified string is then sent to a 'readsf~' which will notify when the file has been read. The delay between the next batch-number to be read is specified by the user. A new file is written simultaneously using the name specified by the user. The delay between read files is necessary so that the 'writesf~' can capture any decay from processes before the next recording commences.

Problems and solutions

The main problem during the design was how to find the split locations for the different sections. The method used was to split the string into individual characters then to search through this string, using 'drip', to separate the floats from the symbols. Each character was also counted, this count number is the location of each character. If the user specifies the format (005) (three digits) then the point where three digits are found in the sequence is recorded. [3] It was decided to reverse the list before counting so that it would begin at the end of the string in case of the following occurrence:

/usr/003harry/files/sound006.wav

If this example was read front to back then the split point would be found at '003{split-point}harry'. If the string is reversed then the split point between batch number and file name is in the correct place. It appeared to be a problem that if the path:

/usr/harry/files/sound006frog.wav

was entered then the split point for the format would be found at 006{split-point}frog.wav instead of '.wav'. However this is not a problem as it is not the format itself that is actually needed but simply the section succeeding the batch-number. This section is resynthesized with the other parts so that the original 'frog.wav' will be attached to the new batch number, then attached to the original string '/usr...'.

[3] The counter is reset every time there is a symbol. This means that it will only find the point where (using the previous example) three consecutive digits are found.

Conclusion

Batnam achieved its goals and has been used for compositional purposes. However it could be developed further by addressing points such as:

- What should occur when the maximum number is lower than the input minimum number? Intuitively it is felt that the input number should decrease towards the argument number.
- Finding a different method of obtaining the desired string sections. Possibly making use of the 'sprintf' object.
- Creating versions of the patch that have simple audio inputs and outputs as well as using catch and return.
- A user can currently use multiple throw~ objects. This leads on to the possibility of using multiple catch objects. This would mean that there should be an abstraction generation function within batnam that will generate the number of 'writefile' patches specified by the user.
- The ability to switch between automatic and manual reading/writing of files.

3 Birds

Summary

The idea of 'birds' also came out of a compositional need. It is used for generating a set number of bird sounds, the number of generated birds is specified by the user.

Design Features

The main design areas involved in this patch were: the creation of abstractions capable of being automatically generated; designing the sound of these abstractions to be bird-like in character; and creating multi-voice functionality. The individual bird abstractions are capable of receiving data (in this case, a trigger) specified for that unit. This is achieved by ensuring that each had a unique first argument. This argument is used for the contained 'receive' objects. A 'poly' object was designed that will send data to these objects. It will receive a number (the bird I.D) and augment it by its first argument. This means that multiple poly-objects can receive

the same input and be used to send data to various birds simultaneously. Designing bird-like morphologies was achieved by using three different abstractions:

- Tweet. Uses an oscillator which rises from one pitch to another using a line signal.
- Chirp. The line signal is modulated (25hz) and added back to the original signal. This signal is then sent to the pitch of the main oscillator.
- Buzz. This signal is multiplied by various cosine functions.

An 'ADS' envelope is used to control the amplitude and the pitch envelope. A random number is generated for the full duration; the attack and decay are set to 20% of the full duration. Since the full duration is always known there is no need to create a 'release' section of the envelope.

Problems and solutions

Due to the desire that the patch varies in texture a problem was encountered on what the ratio of poly objects to bird objects should be. Using the same amount would result in all bird objects performing on every trigger. Using less poly objects than birds [1] helped to achieve this. It meant that if twenty birds were generated then four poly objects would be created. Any of the existing birds could be played instead of all twenty.
Another problem was ensuring the bird patches operated with audio functionality when generated. This was achieved by first sending a loadbang and turning DSP off for each generated bird patch. When the generation process is complete DSP is resumed.
After creating the various bird patches a problem occurred on how to generate the various patches in the desired weighting. This was achieved by creating a 'rand-sym' abstraction which would choose a random bird symbol. Using an idea stimulated by Johannes Kreidler's probability patch (Kreidler, 2009) a combination of 'random' and '>x' was used to make it possible to achieve the desired weighting between bird objects (See figure 1).

[1] Polyobject amount = birdobject amount/5

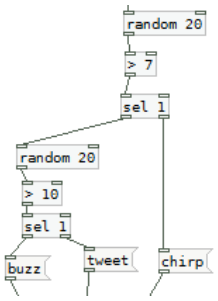
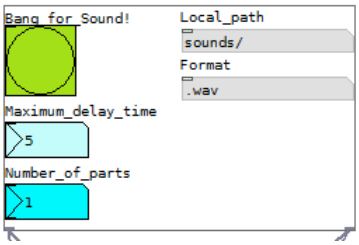


Figure 1: Weighted to favour tweet and buzz over chirp

Conclusion

Many of the aims in this patch were achieved but there is still room for future improvement. It has been used for composition but needed multiple takes due to occasional clicks that occurred when new birds were triggered. This is most likely due to the 'ADS' envelope used. The idea for the envelope stemmed from features found in Miller Puckette's ADSR envelope (Puckette, 2007). Using other features found in Puckette's patch a more flexible envelope could be designed to alleviate this problem. It would also be desirable to produce more variations of the bird abstractions.

4 Compos



Summary

Compos is designed to trigger multiple 'sound-parts' to make a composite whole. The user can determine the number of parts; the maximum amount of delay before playing each part; the local search path for the parts; and the file format to search for.

Design Features

Compos utilises the previous design of automated generation of multiple abstractions, each with unique I.D's. Compos uses a new method for accentuating the volume.

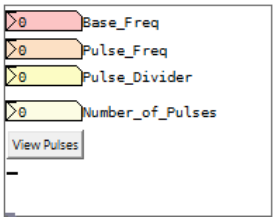
Problems and Solutions

The factors used in accentuating the amplitude are the number of parts and maximum amount of delay. It was decided that, as in birds, the volume should be divided by the maximum amount of parts. The problem with this was the use of random delay meaning that it was highly unlikely that all parts would be played at once. If there are twenty parts, each separate sound would be extremely quite due to the amplitude being divided by twenty. It is necessary to have a decrease in volume according the number of parts but an increase in volume according to an increase in maximum delay. It was decided to have a maximum limit on the division process [2] The maximum delay was used as a factor by using the expression: int($f1/500)/10+1. The result is used to scale the signal. With a maximum limit of 5000 as an input this would result in a maximum scaling of 200%.

Conclusion

Compos is a successful patch that has been used in a compositional context. It could be developed by adding an open-panel dialogue allowing the user to choose any folder on their system.

5 Pulse



Summary

Pulse was originally intended to mimic Csound's 'Fof' formant generator. It was desired for it to be possible to manipulate the individual pulses. Using the idea of clipping parts of a phasor (Puckette, 2007) and having these clipped section enter

[2] A maximum of fifteen. So even if 300 parts are created they will all be divide by fifteen.

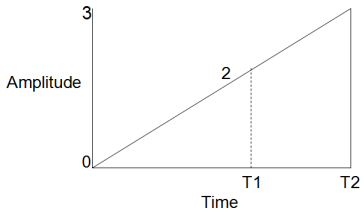


Figure 2: *A phasor with an amplitude of 0 to 3 and temporal length of 0 to T2*

various functions it was possible to create separate pulses. These pulses are then individually modifiable by the user.

Design features

One of the main elements in this design is making use of a single phasor (the master phasor) to generate the multiple separate pulses. The pulses are played sequentially in a group with the phasors amplitude determining the amount of silence between the reoccurring group. This patch, again, made use of the automated abstraction-generation scheme. In this patch, the complications arose with the need for one generated abstraction to pass on information to the next. It also needed a feature to allow users to customise the abstractions.

Problems and Solutions

The pulse abstractions use different parts of the phasor's amplitude. The first pulse will only use the 0-1 section, the second 2-3 ... etc. This would result in the second pulse being generated after the first. A problem occurred due to the desire that the length in time of the pulses be independent from the number of pulses. If the phasor is running at 0-3 in amplitude in time 0 to time T2 then at time T1 the phasor has an amplitude of 2 (See figure 2). If the frequency (length in time) is constant and the phasor is increased in amplitude then the amplitude sections of the phasor will of course become shorter in length (See figure 3). The length must be increased along with the amplitude to maintain a constant space between the amplitude sections. This was achieved by dividing the frequency of the phasor by its amplitude. This ensured that as the amplitude increased the phasor would increase in temporal length.

Each pulse must use a different section of the phasors amplitude. Designing one abstraction to fulfil this was challenging. The main function in the patch is the conditional expr$\sim$ stating that: if the received signal (the phasor) is less than 1 then outlet the signal otherwise a zero. The reverse of this will send the remains of the signal to 's–$1' where $1 is the patches first argument. Each pulse contains a receive object set to receive from ($1 minus 1, the previous pulse). The master

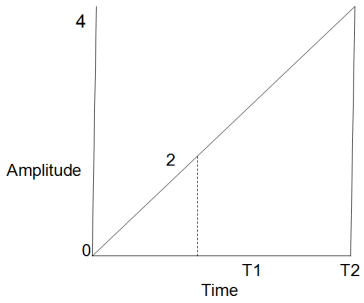


Figure 3: *A phasor with an amplitude of 0 to 4 and the same temporal length of 0 to T2*

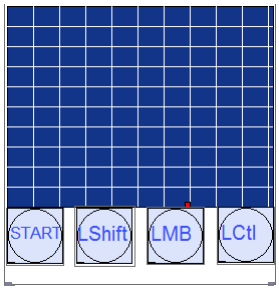
phasor was set to send on 's–0' which results in the first pulse receiving the master phasor, using only 0–1 of this signal, and sending the signal 1–n to the next pulse. The signal is subtracted by 1 to leave 0–(n-1). In the last pulse the received signal will be exactly 0-1. This design meant it was possible to use only the one abstraction for these tasks.

Conclusion

The main aim of this patch having separate pulses was met. It has been very interesting to play with and explore the sounds it can create. However, there is still space for improvement:

- A solo function for the edit pulses dialogue. This could simple turn off all of the pulses except the selected one.
- An ordering of the various sound functions. For instance, all sounds functions within the range of 16-20 would be of a similar type
- The ability to increase the spacing between the individual pulses instead of the spacing between the whole group. This would involve reviewing the key elements of the patch that segregate the phasor. It could be done by explicitly stating which parts of the phasor are read.
- Increasing the functions used
- Introduce a stereo feature with the ability to place pulses in static, or even dynamic, space.

6 Fof



Summary

The aim of this patch was to experiment with sending data to the csoundapi object. The end result is an entertaining patch making use of the mouse and keyboard controls to manipulate various parameters in a 'fof' csound opcode.

Design features

This was the first patch designed and holds little in common with the other creations. The main design features arise from the utilisation of the mouse to interact with the patch. Using gate type methods to turn on/off control data according to which keyboard keys are pressed alongside the left mouse button.

Problems and solutions

The main problem that arose was from the idea of controlling the amplitude using mouse gestures. The aim was that if the mouse is at rest the volume would be zero. If it moved a slight amount then this would be reflected in a slight increase in amplitude. If it moved a great amount this would be reflected accordingly. This was an attempt at simulating Max/MSP's 'delta' output in its mousestate object. One solution was to use a metro object that would take snapshots of the mouses 'y' position once every 'n' ms. The absolute delta value was found between every two snapshots. However, this meant that the amplitude would only increase from zero in very small amounts as the difference between two snapshots would be negligible. A feature was implemented were the input values were recorded every 250ms. If two succeeding values are the same (i.e. if the mouse is at rest) then this value is used as a reference point. The absolute difference of the y values and the reference point would be continuously output to amplitude controls. If the mouse is at rest for another 250ms then this y value will be the new reference point.

Conclusion

This patch was successful in that it achieved its aim. It is also an entertaining patch to utilise. However, the csoundapi object is not entirely stable and at times does not play a score when a message is received. This and other problems could be alleviated by using the OSC protocol to communicate between csound and pure data instead of the csoundapi. However, using the csoundapi has the advantage of being a self-contained patch.

7 Concluding remarks

Overall there has been a clear progression from the first patch (Fof) to the final patch (Pulses). The main idea developed has been of the automatic generation of abstractions. The first stage of this can be noticed in the design of batnam which was created for the purpose of multiple instances. The next stage, in birds, with the abstraction calling and triggering varied objects. Compos uses much of the same idea however instead of simple triggers it is now strings that are input. In pulse the design is at its most complex with audio signals being sent from one abstraction to another and with the ability of modifying the arguments of the pulse objects.

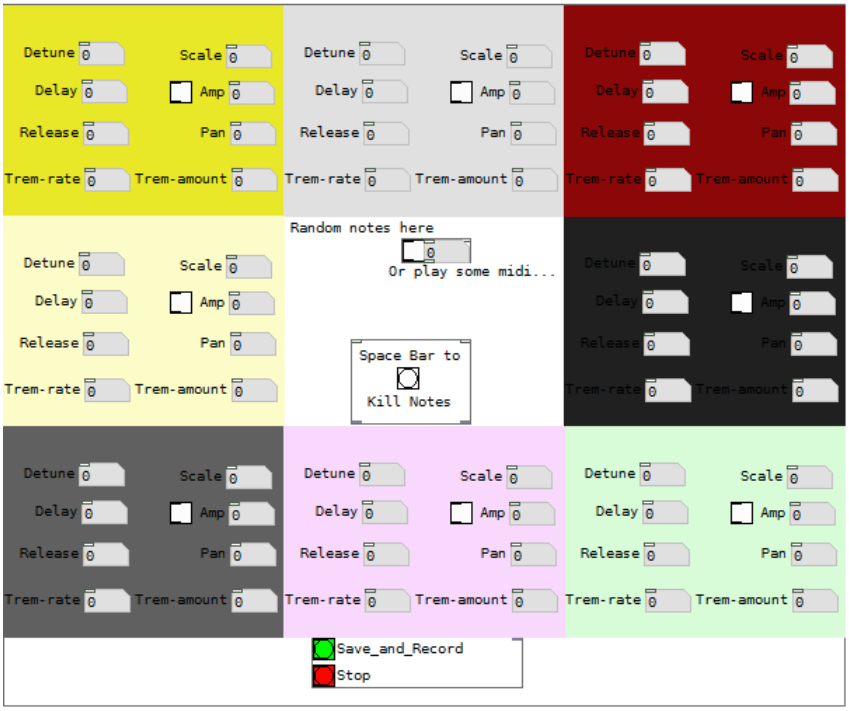
There is still much to be improved upon and learnt about Pure Data and digital audio. Especially in the areas of: communicating data between other software; problem partitioning; and increasing the functionality of multiple instance patches.

References

Kreidler, J. (2009). *Loadbang: Programming Electronic Music in Pd*. Wolke Publishing House.

Puckette, M. (2007). *The theory and technique of electronic music*. World Scientific Pub Co Inc.

A Appendix A: Bell-Synth



A.1 Summary

After trying the 'Fof' patch on a number of systems it was found that not all recognise the csoundapi~ file used by Pure Data. It was felt that another patch should be created to make up for this.

Bell Synth is a polyphonic synthesizer. The user has the freedom to manipulate various parameters of the 8-voices [3] it contains. The parameters are:

- Amplitude
- Relative scaled pitch (relative to root, a scale of *2 will double the pitch. This can be used to add harmonics)
- Detuning in absolute frequencies
- Delay after triggered
- Relative release decay. This is the relative length in time it will take for the voice to decay.

[3] In this description the word voice means a single oscillator. The word voice-bank means a group of 8 of these. The voice-bank will accept a note and velocity value to send to all 8 of its voices

- Pan
- Tremolo rate
- Tremolo amount

A.2 Design and Features

One of the main features in Bell Synth is the 'ADSR' envelope. The design of this came from seminar sessions and also from analysing Miller Puckettes ADSR envelope. The left inlet takes a number or a bang. This number is the peak amplitude that the attack will reach (using values between 0 and 1), a bang will send a 1. If the third argument is not 0, then the second inlet will receive a bang to set the release stage. If the third argument is 0 then the second inlet is not used. Instead when the envelope is triggered, it use the first argument as its sustain before releasing automatically. The second argument is the length of the release relative to the attack/decay lengths. These lengths are determined by the first inlet (velocity). Using the expression (150-($f1*150))+50, where $f1 is a velocity of 0 to 1, then the attack and decay will have a length of 50ms from an input of 1. At near 0 they will have a length of 200ms. The decay amplitude value is the input value multiplied by 0.7. The expression ($f1*150)+50 'inverts' the attack and decay lengths for the release value. This is scaled by the second argument, and sent to the release message. This inversion is done so that a high velocity will have a short attack but a long release and vice versa. The ADSR used originally in Bell-Synth followed this design in MIDI mode. Originally the parent patches fourth argument would supply the release length. Later was customised by adding a third (middle) inlet to dynamically control the release length. The ADSR is sent MIDI velocity numbers that have been scaled accordingly. Using conditional expressions, the value is sent to the left ADSR inlet if >0 and to the right (release trigger) if 0. It was decided for simplicities sake that each voice would only accept one MIDI note and velocity pair and then will disregard all other MIDI notes and velocities until the same MIDI note with a zero velocity is received. In the case of a missing note-off the user can press the space bar (or click the GUI bang) to send the expected MIDI notes+v0 to all voices. In practice these conditions were quite complex to build but it does mean that there is a level of organisation in in a complex MIDI stream. This filtering process is split into two stages. First, using a combination of''onebang' and '==', the first note value received is let through the onebang. It will only let through another note if it equals the first note. If it does then it will send any velocity value paired with this note to the next stage. The second stage, using a similar process, uses first the velocity value that will enter the onebang. If the next value it receives is a 0 (only received from the note stage if it is the correct note pair) then it will send the 0 along with the note number. When a zero is received both stages onebangs are reset.

A.3 Problems and Solutions

It was found that, in this patch, the resetting of the onebangs needed a delay of 1ms to be reset correctly. This is a minor inconvenience that has not caused any problems. Polyphonic functionality is supported by using a 'poly' object with 6 voice-banks to allocate to. The poly object is extremely helpful in that it will send midi/velocity pairs to the correct voice-banks. So it will send 60+127 to voice-bank 1. Then other notes to other voice-banks until a 60+0 is found which will be allocated to voice-bank 1. The voice stealing option in poly was not activated partly due to the goals of the patch, and also due to technical restrictions. It was felt not to include voice stealing so that the first notes played would not be deactivated unintentionally by playing more notes than there are voice-banks Instead these later notes are not activated. The technical problem occurred when experimenting the voice-stealing. The poly object will send a note-off message if a voice is 'stolen'. But in the voice-allocation section this note-off was not registered. This is perhaps due to the speed between note-off and the new note (an issue with speed was found with onebang, see above). This could perhaps be alleviated by delaying the messages a short amount. The tremolo involved a certain amount of experimentation. It was desired that the user could control the depth of the tremolo along with simply controlling its rate. Originally an absolute valued oscillator sent through a quadratic function (for a smoother curved output) which was then scaled by a user set value. This was then subtracted from the original signal. However, this subtraction caused distortion when the user scaled over 50% due instances of, for instance, a negative value of -1 in the original signal being subtracted by an absolute value of 1 from the tremolo oscillator. Instead an out of phase (*-1) version of the original signal was modulated by the oscillator. This result was then scaled by the user and added to the original signal.

A.4 Conclusion

Overall this patch has been very successful and was created in a short length of time. There is as always room for improvement:

- A feature to toggle voice-stealing. Due to the problems encountered with voice-stealing (see above) probably being due to the voice-allocation abstraction . It would be best to not use the voice-allocation abstraction as all the relevant values are sent to the correct voice-banks in poly.

- The ADSR could be simplified. Instead of the first argument being the sustain and 3rd being the mode (MIDI mode: right inlet to set release if >0, or Set mode: the release being triggered after the first argument (in ms)) it could be simplified to: first argument being the length of release and the second

argument being the mode. If the second is 0 then it will be in MIDI mode, if anything above 0 then this value will be used for the sustain in Set mode.

- At the moment the Pan setting is pre-tremolo. Meaning that for every oscillator (48 in total. 8 voices in every 6 voice banks) the tremolo is performed on the left and right channels. It would be more efficient to have these tremolos pre-pan.
- An ability for the user to record parameters and morph between these.
- Varied functions available for the oscillators
- Possibly vibrato options
- Implementation of breakpoint tables for parameters
- Using the mouse-gui on a grid (as in 'Fof') for interaction of certain parameters
- A low-pass filter triggered by velocity values. Low values will decrease the high frequency content and vice versa.
- Tremolo to fade-in after a set or customisable amount of sustain