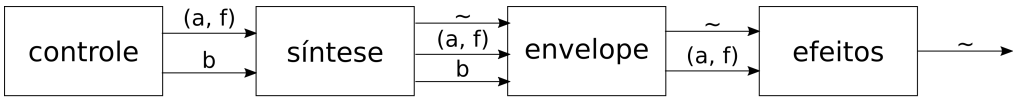
EP3 de Computação Musical – Entrega: 08/07/2017 até 23:55 pelo PACA

Sintetizadores Modulares

Entrega: 08/07/2017 até 23:55 pelo PACA

1 Introdução

Neste exercício-programa iremos implementar um sintetizador modular[1] utilizando PureData. Existem diversos tipos de módulos que podem ser interconectados para constituir um sintetizador, tais como módulos de controle, de síntese, geradores de envelope, filtros e efeitos. O diagrama abaixo servirá de estrutura geral para o EP3; cada módulo será um *subpatch* e cada *subpatch* conterá técnicas alternativas para a especialização do módulo.



Legenda: a = amplitude; f = frequência; b = bang; ~ = sinal.

O trabalho poderá ser feito em duplas e 12 técnicas deverão ser implementadas no total, divididas entre os diversos tipos de módulos assim: 2 modos de controle, 4 técnicas de síntese, 2 tipos de envelope e 4 efeitos. Neste enunciado algumas técnicas serão identificadas como obrigatórias (as mesmas para todas as duplas) e as outras deverão ser selecionadas para preencher as quantidades acima. Nas seções seguintes apresentaremos detalhes sobre os módulos e exemplos de implementações em PureData para diversas técnicas[2]. Exemplos de outros sites podem ser alterados e incorporados, sempre citando as fontes utilizadas (para não configurar plágio) e buscando personalizar as implementações para melhor adaptá-las ao EP. Também faremos referências constantes aos livros *Introduction to Computer Music* de N. Collins, *Elements of Computer Music* de F. R. Moore e *Theory and Techniques of Electronic Music* de M. Puckette, além dos *patches* que vêm junto com a documentação do PureData (acessíveis dentro do Pd por Ajuda->Navegador->PureData).

Neste EP iremos tomar um cuidado especial com a interação entre o usuário e o sintetizador, apresentando (obrigatoriamente) todas as interfaces de controle no *patch* principal utilizando a opção "graph-on-parent"[3] (e mantendo escondidos todos os objetos que não realizam interação com o usuário), no estilo deste sintetizador modular: `https://www.automatonism.com/` (veja qualquer vídeo de demonstração). Tenha em mente que a qualidade da interface gráfica será parte da avaliação. Além dos objetos gráficos tradicionais do pd-vanilla (array, bang, toggle, slider, radio, etc), podemos utilizar knobs e outros objetos da biblioteca *flatgui*, VU-meters e outros objetos da *iemgui* ou desenhar nossos próprios objetos de interação com canvas[4].

[1] `https://en.wikipedia.org/wiki/Modular_synthesizer`

[2] Extraídos principalmente dos sites `http://msp.ucsd.edu/techniques.htm`, `http://pd-tutorial.com/english/index.html`, e `http://write.flossmanuals.net/pure-data/introduction2/`.

[3] `http://write.flossmanuals.net/pure-data/graph-on-parent/`

[4] `http://www.pd-tutorial.com/english/ch05s02.html`

Finalmente, cada implementação de módulo pode ser utilizada várias vezes pelo usuário. Por isso, todas as variáveis identificadas (e.g. vetores, sends/receives, etc.) devem usar o parâmetro $ 0 para distinguir instâncias distintas do mesmo objeto.

2 Controle

O módulo de controle não tem nenhuma entrada pré-definida, mas deve fornecer ao usuário um mecanismo para produzir eventos de controle do sintetizador. A saída dele será feita por 2 `[outlet]`: o primeiro produzirá, a cada início de evento, um par de valores associados à frequência e à amplitude do evento a ser gerado; o segundo sinalizará o fim do evento através de um bang. Estes dados serão utilizados pelos módulos de síntese e de geração de envelopes.

2.1 Leitor de arquivos MIDI (obrigatório)

Uma maneira simples de obter eventos de controle é fazer o sequenciamento de eventos provenientes de um arquivo MIDI. Você deverá fazer o *parsing* desses eventos e o agendamento das informações relevantes para os outros módulos. Alguns objetos que podem auxiliar aqui são `[openpanel]`, `[cyclone/seq]` e `[cyclone/midiparse]`. Um exemplo da utilização conjunta desses objetos pode ser encontrado em `http://paca.ime.usp.br/mod/resource/view.php?id=36257`.

Convém lembrar que o protocolo MIDI prevê eventos distintos NOTE_ON e NOTE_OFF para tratar início e fim de uma nota, respectivamente, porém, o NOTE_OFF é frequentemente substituído por um NOTE_ON com intensidade 0 (zero), o que pode ocorrer tanto em arquivos MIDI quando em dispositivos MIDI de controle.

Diversos arquivos MIDI podem ser baixados nos seguintes websites: `http://imslp.org/`, `http://www.mutopiaproject.org/` e `https://musescore.com/sheetmusic`.

2.2 Entrada MIDI

Outra maneira de obter eventos de controle envolve o uso de dispositivos MIDI[5] em tempo real. Alguns objetos que podem auxiliar aqui são `[notein]`, `[stripnote]` e `[mtof]`, além do *patch* `2.control.examples/17.PART3.midi`. Veja também estes links[6] para a configuração e uso de dispositivos de entrada MIDI em Pd.

2.3 Sequenciador de N-passos

Uma terceira maneira de obter eventos para controle é sequenciando um conjunto de eventos (notas), cada um deles definido por uma tripla de valores (frequência,duração relativa,intensidade), e repetindo-os em

[5] Quem não tiver um dispositivo MIDI físico poderá utilizar um dispositivo virtual (por exemplo, quem usa um sistema Linux poderá utilizar o vkeybd).

[6] `http://write.flossmanuals.net/pure-data/using-midi/` e `http://msp.ucsd.edu/techniques/latest/book-html/node56.html`

um loop com um controle adicional de velocidade (andamento). As durações serão expressas em unidades relativas (beats ou pulsos) e a velocidade em beats por minutos. O usuário deverá poder definir dinamicamente os parâmetros dos eventos e também controlar dinamicamente a velocidade de execução. Um exemplo de sequenciador pode ser encontrado no *patch* `3.audio.examples/C08.analog.sequencer.pd` e também nesse link[7].

2.4 Theremin canvas

Esse modo de interação permitirá ao usuário interagir com o sintetizador através de cliques do mouse em um canvas onde cada ponto (x,y) representa um par (amplitude,frequência)[8]. Os pixels do canvas deverão mapear os respectivos espaços de modo logarítmico: em um canvas de altura M e largura N o pixel (x,y) representará uma amplitude de $10^{\frac{10(x-N)}{N}}$ e uma frequência de $20000 * 10^{\frac{3(y-M)}{M}}$. Cada clique com o botão da esquerda determinará um início de evento com os parâmetros selecionados, e cada clique com o botão da direita representará o final do evento; um clique com o botão da esquerda durante um evento que não terminou deve gerar uma "rampa" com uma sequencia de novos pares de amplitude e frequência, interpolando os pixels (x,y) entre a posição inicial (x0,y0) e a nova posição (x1,y1), durante um tempo determinado por um parâmetro de entrada (default 1 segundo).

3 Síntese

O módulo de síntese receberá através de 2 inlets as informações produzidas por um módulo de controle (par amplitude-frequência e bang de final do evento) e irá gerar 3 informações na saída: o primeiro `[outlet~]` envia o sinal de áudio sintetizado, o segundo `[outlet]` repassa o par amplitude-frequência a cada início de evento (para ser usado pelo módulo de envelope), e o terceiro `[outlet]` repassa o bang a cada final de evento. A interface deste módulo deverá conter um seletor para a técnica desejada, além dos controles dos parâmetros de síntese.

3.1 Gerador de formas de onda (obrigatório)

Módulos do tipo VCO (Voltage Controlled Oscillator[9]) são essenciais em qualquer sintetizador, e geram ondas de formatos diversos, como senoidal, triangular, dente-de-serra, quadrada ou pulso. Esse módulo deve implementar um gerador dessas formas de onda que seja *livre de alias*, ou seja, que use para cada forma de onda o maior número possível de harmônicos cabível dentro da representação. Seu *patch* deve exibir para o usuário a forma de onda sintetizada através de um gráfico. Você pode se inspirar no exemplo desenvolvido em aula, mas deve criar para cada evento (com frequência f_0) uma mensagem personalizada para iniciar a tabela (usando $\frac{22050}{f_0}$ harmônicos com os pesos e sinais apropriados para cada tipo de onda).

[7] `http://write.flossmanuals.net/pure-data/step-sequencer/`

[8] Veja por exemplo `http://www.pd-tutorial.com/english/ch05s02.html`

[9] `https://en.wikipedia.org/wiki/Voltage-controlled_oscillator`

3.2 Síntese Aditiva (Fourier)

Uma maneira de se obter sinais espectralmente ricos consiste em adicionar um grande número de parciais senoidais a partir de uma lista de valores de amplitude, visualizada à maneira de um espectro de Fourier[10], porém contendo apenas os harmônicos do sinal (múltiplos da frequência fundamental). Nesse caso, o usuário deve ser capaz de definir a forma gráfica do perfil do espectro harmônico do sinal desejado, e o módulo deve sintetizar o sinal através da sobreposição de todos os parciais prescritos. Veja por exemplo o *patch* `3.audio.examples/D08.table.spectrum.pd`.

3.3 Síntese Aditiva (Chebyshev)

Outra maneira de se obter diversos harmônicos com controle independente de suas amplitudes relativas é usar a composição de um oscilador senoidal com combinações lineares de polinômios de Chebyshev[11] (esse é um caso particular da técnica de waveshaping). Veja por exemplo o *patch* `3.audio.examples/E05.chebychev.pd`. Nesse módulo o usuário deve ser capaz de escolher as amplitudes relativas para cada um dos harmônicos desejados através de uma interface do tipo *line spectrum*, e a síntese será feita por composição de um gerador senoidal básico com a soma ponderada dos polinômios de Chebyshev correspondentes ao perfil espectral desejado.

3.4 Síntese Aditiva por fórmulas fechadas

Um mecanismo computacionalmente muito eficiente de se obter um grande número de parciais, proposto por J. A. Moorer e conhecido como DSF (discrete summation formulas), consiste em usar fórmulas fechadas para a síntese aditiva de um grande número de parciais, como por exemplo a identidade de Lagrange[12]

$$\sum_{k=1}^{n} \sin(k\omega) = \sin\left(\frac{(n+1)\omega}{2}\right) \frac{\sin(n\omega/2)}{\sin(\omega/2)}.$$

Na prática, essa é uma técnica de síntese não-linear, pois se vale de multiplicações de sinais senoidais. Nesse módulo você deve implementar ao menos 4 tipos diferentes de fórmulas fechadas que produzam perfis espectrais diferentes (ou seja, não use fórmulas que apenas troquem senos por cossenos ou tenham simples variações de fase), fornecendo ao usuário os parâmetros das equações correspondentes. Você pode procurar fórmulas DSF na seção 3.4.3 do livro *Elements of Computer Music* de F. R. Moore ou o próprio artigo do J. A. Moorer *The Synthesis of Complex Audio Spectra by Means of Discrete Summation Formulas*.

3.5 Síntese Subtrativa (ruídos coloridos)

A síntese subtrativa parte de um sinal com espectro rico (isto é, que possui energia em muitas frequências) e remodela o espectro através de filtragem. Nesta técnica os valores em Hz provenientes do módulo de controle podem ser utilizados como frequências de corte (ou centrais) dos filtros utilizados.

[10] `http://msp.ucsd.edu/techniques/latest/book-html/node72.html`
[11] `http://msp.ucsd.edu/techniques/latest/book-html/node84.html`
[12] `https://en.wikipedia.org/wiki/List_of_trigonometric_identities`

No caso específico de ruídos coloridos[13], alguns desses são caracterizados por faixas de frequência específicas, enquanto outros são caracterizados pelo decaimento de energia do espectro (spectral roll-off). Implemente ao menos 4 cores de ruídos diferentes nesse módulo.

3.6 Síntese Subtrativa (Formantes)

Outra aplicação de síntese subtrativa corresponde a usar filtros passa-banda (`[bp~]` em PureData) para enfatizar/valorizar determinadas regiões espectrais, a fim de emular formantes existentes em instrumentos musicais e na voz. Para isso você deve oferecer controles de frequência central e largura de banda para cada filtro, e aplicá-los a um sinal harmônico espectralmente rico (use um pulso periódico para isso). Nesse módulo, considere uma interface com 4 formantes. Para gerar vogais, veja a tabela na página 292 do Moore.

3.7 Karplus-Strong

Essa é uma técnica clássica para simulação de cordas pinçadas, que parte de um vetor aleatório com comprimento associado ao período fundamental $\frac{SR}{f_0}$ e fica recirculando esse vetor por um filtro passa-baixa ($y_n = \varrho\frac{x_n+x_{n-1}}{2}$, com $\varrho \in (0, 1]$), o que visa atenuar mais rapidamente o conteúdo de alta frequência do sinal. Ela está descrita na seção 5.2.1 do Collins e 3.4.5 do Moore, e aparece por exemplo no *patch* `3.audio.examples/G04.control.blocksize.pd` da documentação do Pd. Você deve oferecer ao usuário um controle de duração do sinal, que será associado ao fator de atenuação ϱ na equação do filtro.

3.8 Síntese FM

Essa técnica vista em sala de aula compõe dois osciladores senoidais a fim de gerar um espectro com componentes em $f_c \pm n * f_m$ onde f_c é a frequência central ou portadora e f_m é a frequência de modulação (veja por exemplo os *patches* `3.audio.examples/E08.phase.mod.pd` e `3.audio.examples/E09.FM.spectrum.pd`). Nessa implementação você deve oferecer ao usuário os controles de harmonicidade $H = \frac{f_m}{f_c}$ e índice de modulação $I = \frac{a_m}{f_m}$ (onde a_m é a amplitude de modulação), além de uma opção de vincular o controle de amplitude da entrada ao índice de modulação (nesse caso, o índice real usado na síntese deve ser ajustado para $a * I$ onde a é a amplitude do evento sintetizado e I é o índice de modulação definido pelo usuário na interface).

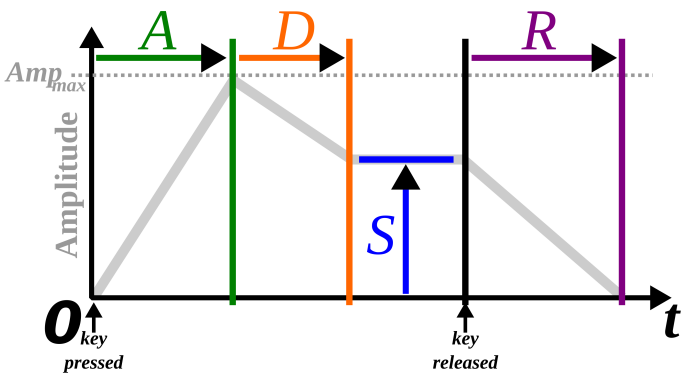
4 Envelope

O módulo gerador de envelope receberá três entradas: o sinal sintetizado, pares de *floats* contendo os valores de amplitude de pico e frequência, e bangs indicando os finais das notas. Sua interface deverá conter um modo de selecionar o tipo de envelope utilizado, além dos parâmetros de controle dos envelopes. A saída será o sinal sintetizado após a aplicação do envelope. Use o objeto `[vline~]` para produzir envelopes e

[13] `https://en.wikipedia.org/wiki/Colors_of_noise`,`http://www.pd-tutorial.com/english/ch03s03.html`

consulte o link `http://write.flossmanuals.net/pure-data/envelope-generator/`. Em todos os casos abaixo seu *subpatch* deve mostrar na interface gráfica o formato do envelope sendo utilizado.

4.1 ADSR (obrigatório)



Neste tipo de envelope teremos 4 fases: ataque (atack), decaimento (decay), sustentação (sustain) e soltura (release) (veja o *patch* `3.audio.examples/D02.adsr.pd`). O usuário deverá ter a possibilidade de determinar os tempos de ataque, decaimento e soltura. O tempo de sustentação não será determinado explicitamente, mas essa fase deve ser mantida até o momento em que chegar o bang relativo ao final da nota. Em relação aos valores de amplitude, o valor máximo de ataque será definido pelo float recebido na entrada, e a amplitude de sustentação será definida por um valor percentual através da interface gráfica.

4.2 Envelope percussivo (AD)

Instrumentos percussivos produzem notas cujos envelopes são bem característicos. O envelope percussivo apresenta um ataque curto seguido de um decaimento exponencial da forma $2^{-\beta t}$, controlado por um parâmetro $\beta > 0$ (quanto maior o β, mais rápido o decaimento). Você pode implementar esse decaimento através de um filtro recursivo com equação $y_n = x_n + 2^{-\beta/SR} y_{n-1}$ alimentado com um pulso unitário ($x_0 = 1, x_n = 0,\ \forall n > 0$). Esse módulo também deve ter um parâmetro para o tempo de soltura, para controlar a finalização do evento após receber o bang sinalizando seu término.

4.3 AD invertido

Uma subversão do envelope percussivo, utilizado com alguma frequência em música eletrônica, é o envelope que possui uma rampa suave de ataque, muito lenta e de duração variável conforme o evento, com a soltura na forma de ataque invertido (rampa rápida para 1 seguida de rampa até 0). O ataque lento não poderia seguir uma rampa exponencial por tempo indefinido (já que chegaria inevitavelmente em 1), mas pode ser criado com uma função sigmoidal da forma $e(t) = \frac{1}{1+e^{\alpha(\tau - t)}}$, onde $\tau > 0$ controla o tempo até a curva atingir o ponto de inflexão e o fator $\alpha > 0$ controla a velocidade da inflexão (valores menores produzem inflexões mais suaves). Sua interface deve permitir ao usuário definir os valores de α e τ, bem como o

tempo total das duas rampas acionadas no momento do fim do evento (aloque metade desse parâmetro para a rampa de subida e metade para a rampa de descida).

4.4 Envelope com trechos lineares aleatórios

Outra construção que leva a uma sonoridade essencialmente eletrônica/sintética é um envelope que entre o ataque e a soltura fica oscilando entre valores de amplitude aleatórios. Nessa construção você deve possuir parâmetros para os tempos de ataque e soltura e também para as durações mínima e máxima dos trechos internos aleatórios (cujas amplitudes podem ser livremente sorteadas entre 0 e 1).

5 Efeitos

O último módulo receberá o áudio relativo aos eventos sintetizados e envelopados, e também os pares de valores (amplitude,frequência), e irá aplicar um efeito selecionado. Em todos os efeitos implementados deve existir um controle *dry/wet* que permite ao usuário controlar dinamicamente uma mistura entre o sinal original (*dry*) e o sinal processado pelo efeito (*wet*). Você pode utilizar os parâmetros de frequência e amplitude para tornar alguns dos efeitos abaixo mais interessantes.

5.1 FIR (obrigatório)

Um filtro FIR realiza uma convolução simples do sinal de entrada com uma lista de coeficientes, e pode ser usado para produzir diversos efeitos, desde uma simples filtragem passa-baixa passando por uma equalização genérica até a simulação acústica de uma sala com resposta espacializada binaural, a depender exclusivamente da configuração da lista de coeficientes. Sua implementação pode usar o objeto `[iemlib/FIR~]` e deve permitir ao usuário ler os coeficientes do filtro a partir de um arquivo, e mostrar na interface tanto a lista de coeficientes quanto a resposta em frequência do filtro (que corresponde ao espectro de magnitude). Usem o fórum do PACA para compartilhar arquivos com coeficientes de filtros interessantes.

5.2 Equalizador gráfico+paramétrico

Um equalizador gráfico possui controle de ganho/atenuação para cada faixa de frequência. Um equalizador paramétrico, por outro lado, aceita controles de frequência central, fator de ganho/atenuação e largura de banda para um certo número de faixas de frequência. Sua implementação deve oferecer simultaneamente as duas modalidades de interação: no caso gráfico através de uma interface gráfica desenhável onde cada coluna do gráfico controla um fator de ganho/atenuação, e no caso paramétrico através de 15 controles contínuos (sliders) que afetam 5 faixas de frequência programáveis.

O patch `3.audio.examples/I03.resynthesis.pd` implementa um equalizador gráfico, fazendo o produto do espectro das janelas de entrada pelo padrão definido no equalizador e ressintetizando o espectro obtido. Cada faixa de um equalizador paramétrico pode ser implementada por um filtro `[bp~]`. Sua

implementação deve permitir ao usuário desenhar no gráfico e também mexer nos controles paramétricos, e selecionar através de um `[toggle]` qual dos dois equalizadores deve ser utilizado no processamento.

5.3 Waveshaping

Waveshaping é uma técnica de processamento de sinais que mapeia os valores de amplitude de um sinal x_n de entrada através de um mapeamento ou *waveshaper* $f : [-1, 1] \longmapsto [-1, 1]$, que realiza uma composição $y_n = f(x_n)$. Sua implementação deve oferecer um controle gráfico para especificação da função f, além de ao menos 2 presets para *distortion (overdrive/fuzz)* e *tube amplifier*.

Os seguintes *patches* contém exemplos diversos de waveshaping: `E04.difference.tone.pd`, `E05.chebychev.pd` e `E06.exponential.pd` (todos em `3.audio.examples/`). Um *fuzz* simples pode ser implementado com uma função de *clipping* (veja a Figura 5.6 em `http://msp.ucsd.edu/techniques/v0.08/book-html/node74.html` e uma implementação de *fuzz* em `https://guitarextended.wordpress.com/2011/12/28/simple-fuzz-effect-with-pure-data/`). Uma outra referência sobre este assunto pode ser encontrada em `http://www.rs-met.com/documents/tutorials/Waveshaping.pdf`.

5.4 Wah-Wah/Auto-Wah/Dynamic Wah

O efeito *wah-wah*[14] corresponde a um filtro passa-banda com frequência central variável e largura de banda pequena (ou passa-baixa com frequência de corte variável), cuja saída é mixada ao sinal original. Quando implementado na forma de pedal contínuo, a posição do pedal costuma determinar a frequência central do filtro. Na ausência do pedal, é possível controlar dinamicamente a frequência central de várias maneiras, por exemplo atrelando a frequência central ao envelope de amplitude ou a um oscilador de baixa frequência (LFO – low frequency oscillator), em um efeito conhecido como *auto-wah*[15].

Sua implementação deve permitir o controle dinâmico do efeito, seguindo o envelope de amplitude (por exemplo com `[env~]`) e também de forma periódica usando um LFO (com controles de amplitude e frequência do LFO). O seguinte link traz uma implementação desse último caso utilizando um `[vcf~]`: `https://guitarextended.wordpress.com/2012/01/07/wha-wha-auto-with-pure-data/`.

5.5 Phaser

Outro efeito implementado com filtros variantes no tempo é o *phaser*[16]. Esse efeito consiste na aplicação em série de diversos filtros *notch* (rejeita-banda) com frequências centrais variáveis, e na soma desse resultado com o sinal original. Um efeito perceptualmente parecido pode ser obtido pela aplicação de filtros passa-tudo somados ao sinal original, onde cada filtro é responsável por alterar a fase do sinal e a soma irá gerar diversas interferências (construtivas e destrutivas) criando essencialmente o mesmo efeito.

[14] `https://en.wikipedia.org/wiki/Wah-wah_pedal`

[15] `https://en.wikipedia.org/wiki/Auto-wah`. Note que a nomenclatura pode se tornar confusa pelo uso de termos diferentes por diferentes fabricantes de pedal: a Boss por exemplo denomina o auto-wah com LFO de Auto Wah e o auto-wah com envelope de Dynamic Wah, sendo que esse último pedal possui também o modo LFO.

[16] `https://en.wikipedia.org/wiki/Phaser_(effect)`

O exemplo `3.audio.examples/H15.phaser.pd` apresenta uma possível implementação de um phaser. Sua implementação deve permitir ao usuário determinar a frequência e amplitude de um LFO que controle automaticamente os parâmetros do phaser.

5.6 Flanger

Um efeito interessante e relacionado ao phaser é o *flanger*[17], que consiste em somar o sinal da entrada a uma cópia com atraso variável (o que em PureData pode ser implementado através de uma *delay line* com o objeto `[vd~]`). Esse tipo de filtro, cuja equação é $y_n = x_n + x_{n-d_n}$ corresponde a um *comb filter* variável, ou seja, a uma série de filtros *notch* (rejeita-banda) equi-espaçados, o que evidencia o parentesco desse efeito com o *phaser*. O link `http://msp.ucsd.edu/techniques/latest/book-html/node119.html` e o patch `3.audio.examples/G05.execution.order.pd` apresentam uma implementação de um *flanger* cujo *delay line* possui atrasos controlados por um LFO. Sua implementação deve oferecer parâmetros de frequência e profundidade controláveis pelo usuário.

5.7 Reverb

Outro efeito relacionado a *delay lines* é o reverberador, como aquele encontrado no exemplo `3.audio.examples/G08.reverb.pd`, que implementa as reflexões com diversos valores de atrasos e simula diferentes tamanhos de sala através dos ganhos no feedback dos filtros de atraso. Uma alternativa a esse reverberador simples é utilizar o objeto `[freeverb~]`, que possui diversos argumentos para controlar aspectros diversos da simulação acústica. Sua implementação deve possuir controles que permitam ao usuário obter diversos tipos de reverberação (os controles específicos podem depender da implementação).

6 Entrega

Deverá ser entregue pelo Paca uma biblioteca de arquivos .pd contendo a implementação de cada módulo (em um arquivo separado), um arquivo principal ilustrando um possível uso do sintetizador modular, e um relatório sucinto descrevendo as escolhas das técnicas, as fontes/referências/patches utilizados e as modificações realizadas. Não esqueça de identificar na implementação e no relatório os nomes completos e números USP dos integrantes da dupla. Todas as implementações deverão estar fartamente documentadas e visualmente organizadas: use modularização para evitar subpatches que não caibam na tela, `[trigger]`s para sequenciar claramente as ações no patch, e pares `[send]`/`[receive]` para evitar patches "emaranhados".

Nas duas últimas aulas do curso (4 e 6 de julho) cada dupla terá 10 minutos para apresentar em sala de aula seus sintetizadores, a fim de mostrar o resultado das explorações sonoras realizadas.

Boas implementações! =)

[17] `https://en.wikipedia.org/wiki/Flanging`