**UNIVERSIDADE ESTADUAL DE MARINGÁ**
**PROGRAMA DE INICIAÇÃO CIENTÍFICA – PIC**
**DEPARTAMENTO DE MÚSICA**
**ORIENTADOR: PROF.° DR. MARCUS ALESSI BITTENCOURT**
**ACADÊMICO: MAURÍCIO PEREZ**

# CRIAÇÃO DE UM MODELO COMPUTACIONAL DE UM SINTETIZADOR MODULAR ANALÓGICO NO AMBIENTE DE PROGRAMAÇÃO VISUAL PURE DATA

**Maringá, 30 de agosto de 2010**

**UNIVERSIDADE ESTADUAL DE MARINGÁ**
**PROGRAMA DE INICIAÇÃO CIENTÍFICA – PIC**
**DEPARTAMENTO DE MÚSICA**
**ORIENTADOR: PROF.° DR. MARCUS ALESSI BITTENCOURT**
**ACADÊMICO: MAURÍCIO PEREZ**

# CRIAÇÃO DE UM MODELO COMPUTACIONAL DE UM SINTETIZADOR MODULAR ANALÓGICO NO AMBIENTE DE PROGRAMAÇÃO VISUAL PURE DATA

**Relatório contendo os resultados finais do projeto de iniciação científica vinculado ao PIC – UEM**

**Maringá, 30 de agosto de 2010.**

**Sumário**

## 1 – Resumo

Este projeto propôs a confecção de uma simulação virtual computacional de um sintetizador analógico modular, realizada utilizando-se a linguagem de programação visual Pure Data, com a implementação de diversos módulos comuns presentes nos sintetizadores modulares. O trabalho envolveu o estudo das técnicas clássicas de síntese sonora e o estudo das técnicas de programação visual com Pure Data e suas aplicações na implementação das técnicas de síntese clássicas. No processo de estudo e pesquisa, foi produzido material bibliográfico instrucional para o website wiki de documentação do LAPPSO-UEM.

**Palavras chaves:** Computação Musical, Pure Data, Sintetizador Modular.

## 2 – Introdução

Este projeto propôs a confecção de uma simulação virtual computacional de um sintetizador analógico modular. Esta modelagem computacional foi realizada utilizando-se a linguagem de programação visual Pure Data (ver PUCKETTE 2007).

O sintetizador analógico modular (ver CHADABE, 1997) consiste em uma montagem de diversas unidades eletrônicas de processamento de sinais de áudio chamadas de módulos que, quando interconectadas por meio de fios curtos (patch cords) em uma configuração determinada formam um circuito de processamento que produz sons e timbres musicais específicos. A combinação dos sinais gerados pelos diversos módulos em diversas configurações pode potencialmente produzir um número infinito de tipos diferentes de sons e timbres. Entre os mais típicos módulos disponíveis para a montagem de um sintetizador modular estão o VCO (Voltage Controlled Oscillator), que produz um som de altura definida, quer como onda senoidal, quadrada, triangular ou dente-de-serra, o NOISE, produz ruídos sem altura definida tais como ruídos branco e rosa, o VCF (Voltage Controlled Filter), uma implementação eletrônica de filtros passa-alta (high-pass), passa-baixa (low-pass) e passa-banda (band-pass), que servem para atenuar ou amplificar zonas específicas de freqüências presentes no áudio que serve de input, o VCA (Voltage Controlled Amplifier), que serve para variar a amplitude do sinal que entra, em resposta a uma variação de voltagem usada como controle, o EG (Envelope Generator), que formata o som com um envelope específico, geralmente baseado em uma configuração ADSR (Attack, Decay, Sustain, Release), o LFO (Low Frequency Oscillator), semelhante ao VCO mas operante abaixo de 20 Hz, que serve

como controle voltáico para outro módulo, e o RM (Ring modulator), que é utilizado para criar sinais que combinam a soma e a subtração de dois sinais originais. Desenvolvidos nas décadas de 60 e 70, os mais famosos sintetizadores analógicos da história são o Moog Synthesizer, criado por Robert Moog em 1963, o Buchla Synthesizer, criado por Don Buchla também em 1963, e o Serge Modular Music System, criado por Serge Tcherepnin na década de 70 (ver CHADABE, 1997). Entre os mais famosos exemplos históricos de uso destes aparatos estão as coleções de álbuns do "Switched-On-Bach" de Wendy Carlos, produzidos nas décadas de 60 e 70 com um sintetizador Moog, e os discos "Silver Apples of the Moon" (1967) e "The Wild Bull" (1968) de Morton Subotnick, produzidos com um sintetizador Buchla. Caríssimos tanto em sua época como na de hoje, estes tipos de sintetizadores podem hoje ser modelados virtualmente em computadores, por meio de softwares também inspirados pelo design modular como PureData ou RTcmix. Pure Data (ou simplesmente Pd) é uma linguagem visual de programação desenvolvida por Miller Puckette nos anos 90 para a criação de trabalhos multimídia e música eletroacústica interativa e em tempo-real (ver Puckette 2007). O Pd é um projeto de software-livre e tem uma imensa base de desenvolvedores que continuamente adicionam novas extensões, funcionalidade e bibliotecas de código ao programa.

Este projeto implementou no ambiente do Pure Data diversos dos módulos comuns presentes nos sintetizadores modulares, conjuntamente com uma montagem destes componentes em um aplicativo útil para os trabalhos de criação musical do Laboratório de Pesquisa e Produção Sonora do Departamento de Música da UEM. O trabalho envolveu o estudo das técnicas clássicas de síntese sonora (aditiva, subtrativa, AM, FM, waveshaping, etc.), guiados pelos trabalhos de Curtis Roads e Herbert Eimert (ver Eimert 1958 e Roads 1996), e o estudo das técnicas de programação visual com Pure Data (ver Puckette 2007) e suas aplicações na implementação das técnicas de síntese clássicas. No processo de estudo e pesquisa, foi produzido material bibliográfico instrucional para o website wiki de documentação do Laboratório de Pesquisa e Produção Sonora do Departamento de Música da UEM (http://www.dmu.uem.br/lappso).

## 3 – Sintetizador modular analógico *versus* sintetizador modular digital

Os sintetizadores modulares analógicos permitiram aos músicos a exploração de uma gama de possibilidades de síntese sonora. O modo de operação por *patch cords* tinha como principais características o resultado sonoro imediato do processo de configuração, em oposição aos primeiros

sintetizadores (RCA Mark II Sound Synthesizer), e a fácil manipulação dos controladores – *knobs*, *switches* e *sliders* – durante a performance.

No entanto, o processo de configuração entre os módulos nestes sintetizadores trazia alguns inconvenientes. Nos sintetizadores modulares analógicos, para a manipulação de cada parâmetro sonoro era necessário um *knob* ou *switch* individual, e para cada conexão entre os módulos era nescessário um *patch cord* exclusivo. No Moog III, por exemplo, criado no começo da década de 1970, havia aproximadamente 150 *knobs* e 35 *switches*, e para a criação de um *patch* – o resultado da configuração entre os módulos por meio dos *patch cords* – poderiam ser usados de 30 a 50 *patch cords*. Além disso, uma vez que se quisesse alterar o *patch* não havia como armazenar o anterior para reutilizá-lo, para isto o músico deveria reconfigurar todo o *patch*, o que poderia levar horas, sem ter a garantia da reprodução idêntica do som originalmente sintetizado (ROADS, 1996).

Por outro lado, no domínio digital, foi possível solucionar os problemas e ir além do paradigma dos sintetizadores analógicos. Em 1957, Max Mathews desenvolve um programa de computador capaz de fazer síntese sonora, o MUSIC-1 (PORRES, 2009). Este programa foi oficialmente a primeira linguagem de Computação Musical e caracteriza-se por ser procedural. O termo refere-se a um paradigma de programação baseado no conceito de chamadas a procedimento. Procedimentos, também conhecidos como métodos ou funções, contém um conjunto de passos computacionais a serem executados a partir de um código [1]. Este código era criado pelo programador a partir de uma interface textual e posteriormente compilado para a linguagem de máquina. O MUSIC-1 teve sua continuidade na série MUSIC-N e influenciou a criação de outras linguagens procedurais de Computação Musical como o Csound (década de 1980) e o Rtcmix (década de 1990).

Em 1990, Miller Puckette, o mesmo criador do Max/MSP, cria o Pure data. O Pure data, ou simplesmente Pd, é um ambiente de programação visual que, diferentemente das formas de programação citadas anteriormente, permite a manipulação do código de maneira gráfica. O Pd é baseado na idéia de "caixas" interconectadas que são geradores de som, processadores de sinal, ou processadores de dados (PORRES, 2009).

A idéia básica é que cada uma destas "caixas" contém um pedaço de código que conforma um pequeno programa que pode ser reutilizado a qualquer momento. Por este motivo, os *softwares* baseados neste tipo de linguagem são chamados de *Patchers*, neologismo originário de *patch* que em inglês significa "pedaço".

Este paradigma nos remete também aos sintetizadores analógicos modulares, dos quais os módulos podem ser representados pelas caixas e os *patch cords* por linhas que conectam estas caixas formando um circuito de processamento. Devido a isto, o Pd é considerado também uma

linguagem de programação orientada a fluxo de informações, mais precisamente, dados.

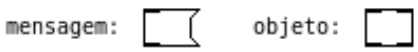
A vantagem da implementação computacional de um sintetizador modular analógico consistiu no fato de que a configuração do circuito de processamento pôde ser otimizada com um único comando e de maneira instantânea, substituindo assim o inconveniente da configuração física por *patch cords*. Como conseqüência, foi possível a pré-configuração e armazenamento de vários *patchs* que podem ser invocados a qualquer momento por meio de *presets*. Outro ponto interessante foi a possibilidade de se manter o uso de controladores na interface digital, por meio de *GUIs* próprios da linguagem de programação Pure data, permitindo a manipulação em tempo real.

## 4 – A linguagem de programação Pure data

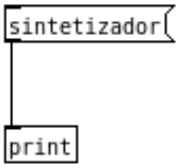
Como todas as linguagens de programação o Pd possui uma sintaxe e uma semântica operacional própria. No caso, como elementos sintáticos o Pd suporta três tipos básicos: *átomos, mensagens e objetos*.

Átomos são os tipos de unidades de dados mais simples no Pd, que podem ser números (1, 10, 33, etc.) ou símbolos, que é tudo aquilo que não é número (x, char, +, &, $, etc). Podem também se apresentar em forma de lista, ou seja, em conjuntos de números (1 45 33) ou símbolos (; bang $ ), ou também em conjuntos de números e símbolos (send $1 +3). Os átomos geralmente são identificados como conteúdo de mensagens ou argumentos de objetos.

A manipulação dos dados é feita por meio das mensagens e dos objetos. Ambos são identificados em formas de caixas:




Basicamente, as operações são feitas da seguinte maneira, as mensagens enviam dados para os objetos e estes, por conseguinte, processam a informação e desempenham alguma função de acordo com o programa contido no objeto. Por exemplo:



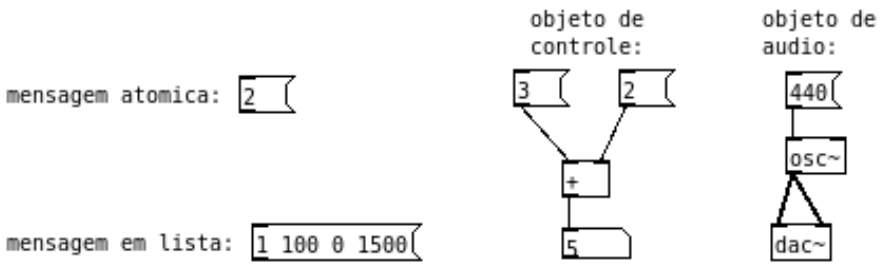
Na figura acima temos o objeto chamado *print*. Segundo o programa deste objeto, qualquer dado que chegar até ele, no caso, a palavra "*sintetizador*" contida na caixa de mensagem, é impresso na tela principal do Pd.

O circuito do fluxo de dados percorre os canais de entrada e saída das mensagens e objetos. Os *inlets* e *outlets*, como são chamados estes canais, são identificados consecutivamente por pequenos retângulos em negrito no topo e na parte inferior das caixas e são interligados por meio de linhas que representam cabos de conexão (*patch cords*).

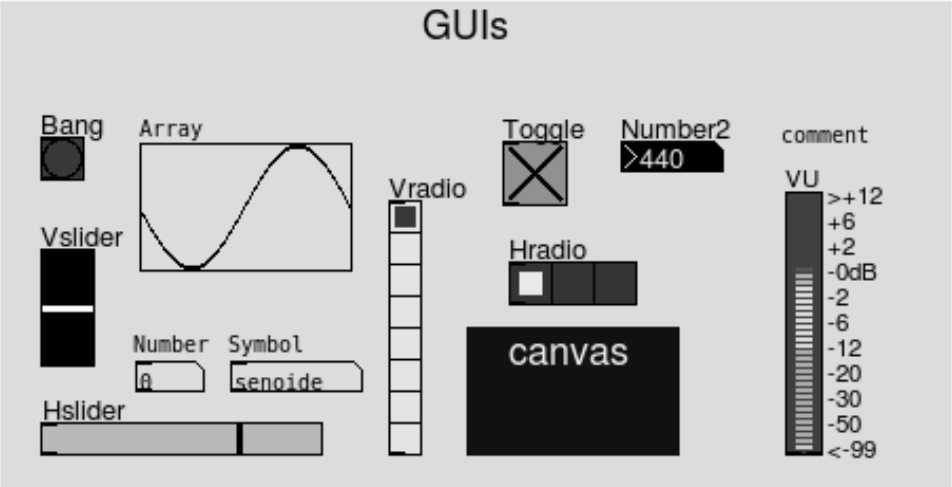

No Pd há tipos específicos de mensagens e objetos. As mensagens são classificadas de acordo com a maneira de como os átomos estão contidos nelas, ou seja, podemos ter mensagens na forma de lista de dados ou na forma de um único dado, quando há apenas um átomo. Neste último caso são chamadas de mensagens atômicas.

Quanto aos objetos podemos classificá-los em *objetos de controle* e *objetos de áudio*. Os objetos de controle caracterizam-se por gerar e processar dados puros (átomos), e agem apenas quando são requisitados(PORRES, 2009).

Os objetos de áudio tem como característica gerar e processar sinal de áudio e, ao contrário dos objetos de controle, esses objetos estão sempre gerando dados. São identificados pelo acréscimo do til (~), que remete o formato de uma onda senoidal, após o nome do objeto e por possuírem linhas mais grossas que indicam o fluxo de sinal de áudio (PORRES, 2009).

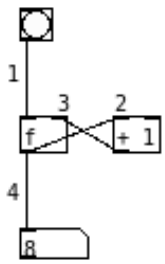


Além das mensagens e objetos o Pd possui também elementos de interface gráfica que podem ser incluídos no *patch*. São os chamados *GUIs* (*Graphical User Interfaces*) que também podem ser utilizados para manipulação de dados, principalmente numéricos, de uma maneira mais interativa e para deixar o *patch* mais "amigável", por meio da edição de cores, tamanhos, etc.

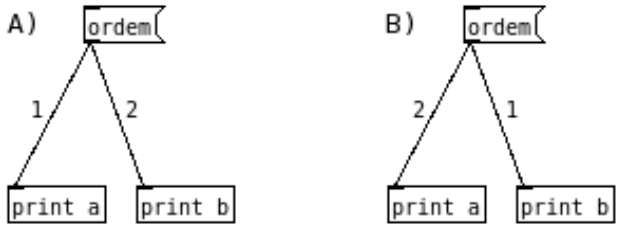



Para que um programa (*patch*) em Pd seja executado de forma satisfatória devemos respeitar determinadas regras de organização das operações. Chamamos isto de Semântica operacional. No Pd devem ser levados em consideração dois aspectos fundamentais: a ordem de execução das *inlets*, a ordem de conexão dos *patch cords*.

A maior parte dos objetos no Pd possuem sua sintaxe operacional padronizada no conceito de *inlets* "quentes" e "frias". Este conceito define que a *inlet* da extrema esquerda do objeto é responsável por faze-lo desempenhar sua função e enviar o resultado por sua *outlet*. Por este caráter executivo, esta *inlet* é chamada de "quente". As demais *inlets* do objeto são chamadas de "frias" pois não executam nenhuma ação (saída de dados), geralmente servem para armazenar informações que alteram os parâmetros do objeto.



A ordem de ocorrência dos eventos é diretamente relacionada à ordem de conexão dos patch cords, logo, ao programar um patch, o "cabeamento" deve ser feito seguindo a lógica do fluxo de informações desejado.




No caso A, a ação a ser executada primeiro será o objeto [print] de variável *a* e depois o objeto [print] de variável *b* (resultado impresso: ordem: a; ordem: b). No caso B o inverso acontece (resultado impresso: ordem: b; ordem: a).

Em decorrência da sintaxe de execução das *inlets* convencionou-se que a ordem dos eventos devem acontecer da direita para a esquerda, sendo que o último *patch cord* deve ser cabeado para a *inlet* "quente".

## 5 – Sintetizador modular digital em Pure data

Neste capítulo iremos tratar da implementação do modelo virtual de um sintetizador modular. Em princípio, por uma questão didática, cada módulo será analisado individualmente, sendo abordado as técnicas clássicas de síntese sonora, as etapas de implementação e a teoria de funcionamento dos *patchs* em Pd. Posteriormente será tratado a maneira pela qual estes módulos se interconectam e sua lógica no circuito de processamento.

Começaremos primeiro pelos módulos geradores de sinal, que são o ponto de partida para a síntese, passando depois pelos processadores de sinal e os controladores.

### OSCILADOR

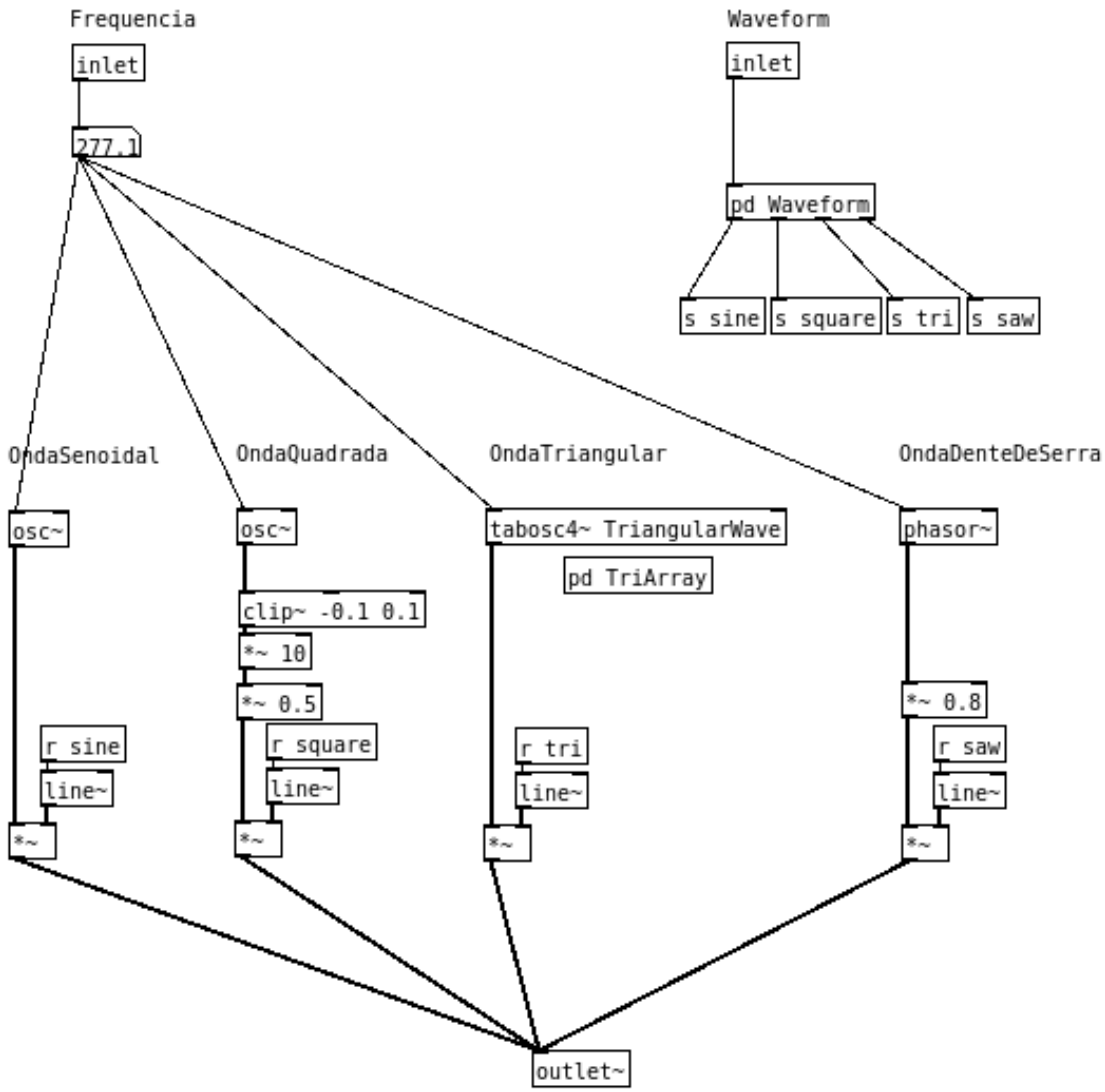
O oscilador é a unidade mais elementar de um sintetizador. Sua funcionalidade básica é gerar uma onda sonora com uma freqüência e uma forma específica (*waveform*). O oscilador digital

opera sob o conceito de *wavetable*. A *wavetable* é uma tabela de valores extraída do cálculo de um ciclo de uma onda sonora. Esta tabela é armazenada na memória do computador e a partir do processamento desta tabela em *loop* é gerada uma onda periódica (ROADS, 1996).

Toda esta unidade foi baseada na criação do objeto chamado [OscCentral~]. Este objeto possui duas *inlets*, sendo, da direita para a esquerda, uma responsável pela seleção do tipo de onda e outra pela definição da freqüência(Hz). Há quatro tipos de onda disponíveis: senoidal, quadrada, triangular e dente-de-serra. Cada uma delas foi confeccionada de acordo com o resultado sonoro almejado, logo, possuem métodos diferentes.

A onda senoidal foi feita com o uso do objeto [osc~], presente na biblioteca nativa do Pd. A onda quadrada com os objetos [osc~] e [clip~], que em sua lógica consistiu no corte das extremidades de uma onda senoidal, buscando ao mesmo tempo o efeito de uma onda quadrada mas com uma pequena atenuação dos cortes, que genericamente seriam cortes retos (KREIDLER, 2009). Para a construção da onda triangular utilizou-se o objeto [tabosc4~]. Este objeto opera sobre a leitura de uma *wavetable* desenhada pelo método *sinesum*, que significa literalmente o resultado da soma das amplitudes das parciais de ondas senoidais. Assim como a onda senoidal a onda dente-de-serra também foi feita pelo uso de outro objeto nativo do Pd, o [phasor~].

Como resultado o objeto [OscCentral~] envia pela sua *outlet* um sinal que contém um determinado tipo de onda e uma freqüência. A freqüência é instanciada nos próprios objetos anteriormente citados, que possuem este método.




*Objeto [OscCentral~]*

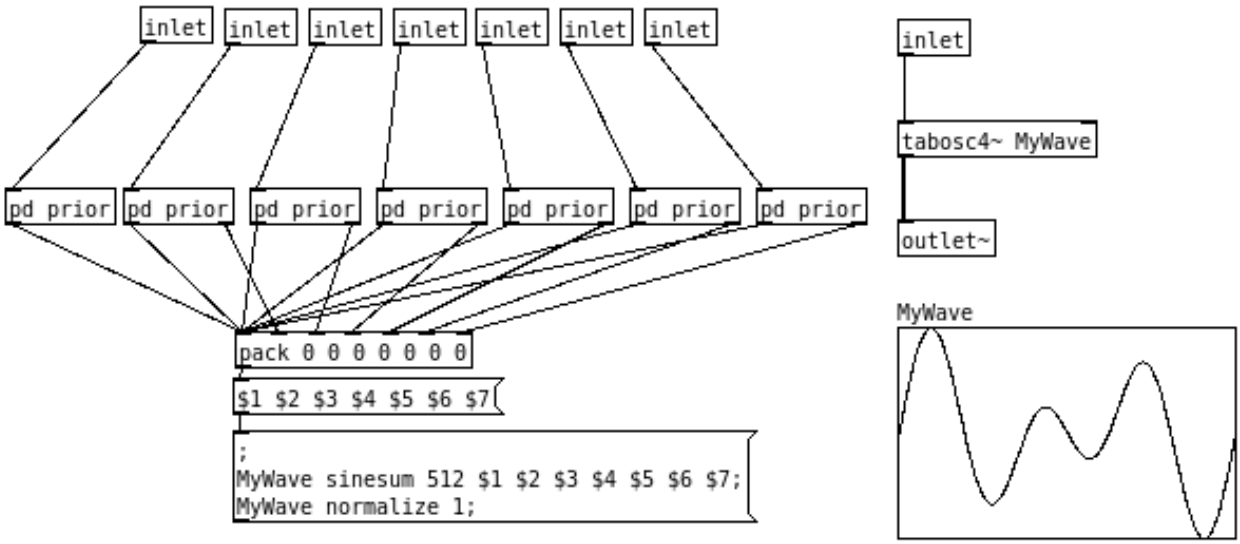
**WAVETABLE**

Esta unidade tem a mesma funcionalidade que o oscilador e gera uma *waveform* com altura definida. No entanto seu diferencial é o controle dinâmico da tabela de valores. Como conseqüência há a possibilidade de gerar um número infinito de formas de onda e, conseqüentemente, timbres. O método por trás desta operação é o da Síntese Aditiva.

Como o próprio nome diz a Síntese Aditiva consiste na soma de ondas sonoras que tem como resultante uma onda complexa (ROADS, 1996). A síntese pode ser feita com a soma de formas de onda diferentes (senoidal + quadrada), com várias ondas de mesma forma (senoidal + senoidal) ou com ambos os métodos. A Síntese Aditiva pode também se apresentar como adição

harmônica ou adição inharmônica. No primeiro caso a soma das ondas se dá em relação aos múltiplos inteiros da primeira parcial (1/1, 1/2, 1/3, 1/4, 1/5, etc.). Já a adição inharmônica não obedece esta lógica na soma das parciais, sendo assim permitido somá-las por qualquer relação intervalar.

Esta unidade foi baseada no objeto [Wavetable~]. Este objeto possui oito *inlets*, sendo, da esquerda para a direita, uma para a entrada da freqüência fundamental e as demais para as intensidades de suas sete primeiras parciais harmônicas, incluindo a fundamental. O núcleo deste objeto foi fundamentado no objeto [tabosc4~] e no método *sinesum*.



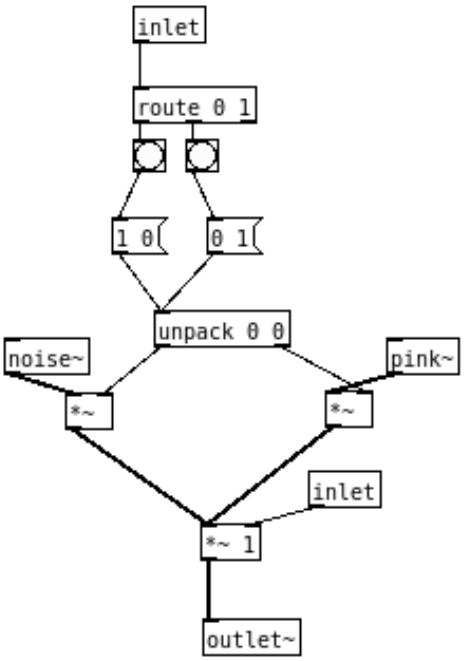

*Objeto [Wavetable~]*

**NOISE**

O módulo NOISE é a última unidade geradora de sinal do sintetizador. Esta unidade foi baseada na construção do objeto [Noise~]. Este objeto possui apenas uma *inlet*, que é responsável pela seleção do tipo de ruído, podendo este ser *white noise* ou *pink noise*. O [noise~] e [pink~] são os objetos nucleares de [Noise~], e pertencem à biblioteca nativa do Pd.

O conceito de ruído (*noise*, em inglês) é compreendido pela coexistência de todas as faixas de freqüência audíveis com máxima amplitude (ROADS, 1996). Digitalmente pode-se conseguir este resultado pela geração randômica de pontos sobre uma tabela. Por padrão esta tabela possui a extensão de -1 a 1 de amplitude e é preenchida por 44.100 pontos. Esta definição cabe precisamente ao ruído branco (*white noise*). O ruído rosa (*pink noise)* porém, é um ruído branco que passa por

cortes das freqüência mais agudas por meio do uso de filtros. O motivo para o uso desta transformação é para uma audibilidade equalizada das freqüências pois, por mais que as freqüências estejam com a mesma intensidade, os sons mais agudos são percebidos pelo ouvido humano com maior "energia", logo se sobrepõem aos graves (KREIDLER, 2009). A linha de corte, ou atenuação, é do decaimento de 3 dB por oitava, ou seja, quanto mais alta a freqüência menor sua amplitude.




*Objeto [Noise~]*

Veremos agora os módulos que são processadores de sinal. Dividiremos estes módulos em *moduladores de sinal* e *"esculpidores" de sinal*.

Chamamos de *moduladores de sinal* aqueles módulos que tem a capacidade de alterar algum aspecto do sinal original a partir do comportamento de um outro sinal (MIRANDA, 2002). Os módulos correspondentes são AM, RM e FM. Estes módulos correspondem às técnicas clássicas de síntese sonora, nas quais o sinal original é chamado de portador (*carrier*) e o sinal que o altera de modulador (*modulator*).

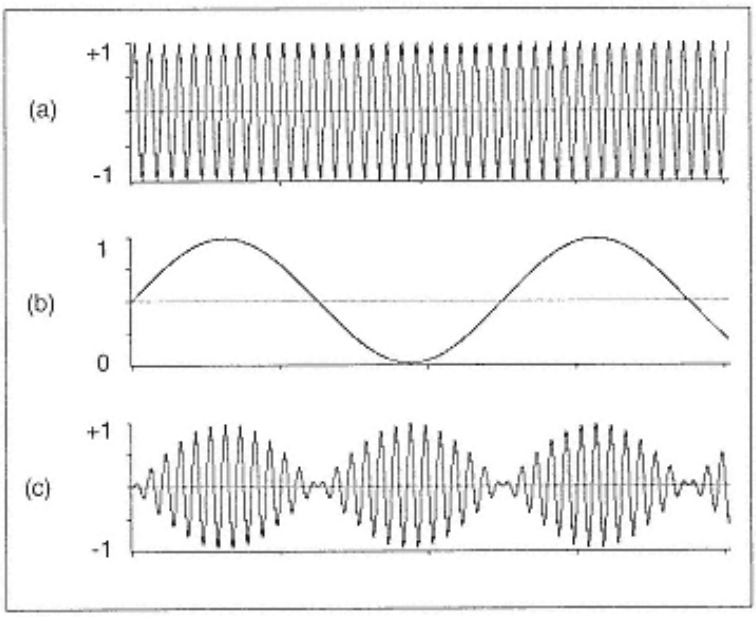
Já os módulos que chamaremos de *esculpidores de sinal* têm a propriedade de atribuir parâmetros ao sinal a fim de dar forma ao mesmo. Os módulos representantes são ADSR, FILTROS, DELAY e REVERB. Podemos ainda distinguir estes módulos por *esculpidores subtrativos*, que são aqueles que decrementam atributos do sinal – ADSR e FILTROS – e *esculpidores aditivos*, que incrementam atributos ao sinal – DELAY e REVERB.

Por último veremos a maneira pelo qual o som resultante do processo da síntese é acionado. Isto é possível por meio de um controlador, no caso, representado pela unidade TECLADO.

**AM**

O módulo AM corresponde à técnica de síntese clássica chamada Modulação de Amplitude (*Amplitude Modulation*).

Como o próprio nome evidencia, a Modulação de Amplitude (AM) é uma técnica na qual o sinal original tem sua amplitude modulada de acordo com um sinal modulador (ROADS, 1996). Neste tipo de modulação a *carrier (a)* possui uma tabela de amplitude de onda de -1 a +1 e a moduladora de 0 a 1*(b)*. Estes dois tipos de sinal são chamados respectivamente de *bipolar* e *unipolar*, e são importantes para entender o que ocorre na síntese AM*(c)*. O que acontece neste tipo de modulação é que o sinal modulador *unipolar* ao ser multiplicado como amplitude da *carrier* serve como um "potenciômetro oscilante" para a mesma. A taxa de oscilação varia de acordo com o período determinado pela freqüência da moduladora.

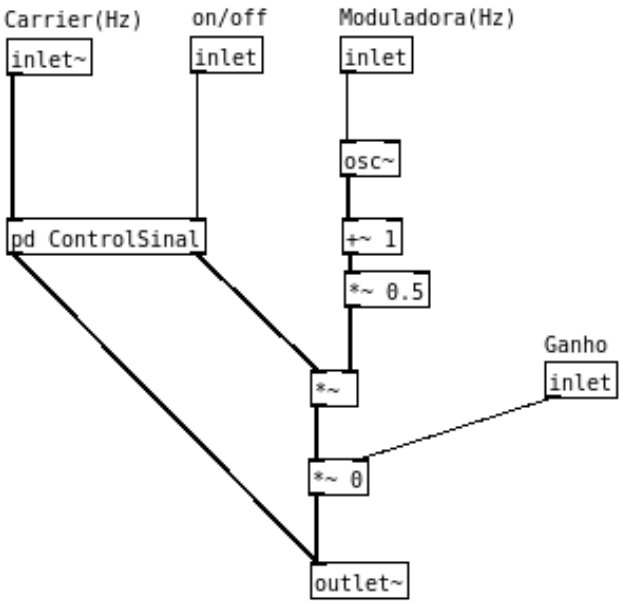



*Sintese AM - figura retirada de ROADS (1996)*

Quando o valor da freqüência do sinal modulador está abaixo de 20Hz (sub-áudio) ouvimos como resultado da modulação um efeito *trêmolo*(variação de intensidade). Ao ultrapassarmos deste limiar são acrescentadas parciais ao espectro do sinal, resultando assim em um timbre (MIRANDA, 2002). Estas parciais são acrescentadas em relação ao sinal original. Para encontra-las basta somar e subtrair a moduladora pela *carrier*. Sendo assim, quando acima de 20Hz, a síntese AM produz duas parciais a mais além do sinal original (ROADS, 1996).

Esta unidade do sintetizador foi baseada na construção do objeto [AM~]. Este objeto possui

quatro *inlets*. Da direita para a esquerda, a primeira é responsável pela entrada de um sinal de áudio (*carrier*). As duas *inlets* conseqüentes são responsáveis respectivamente pelo acionamento da unidade de síntese (*on/off*) e de um possível ajuste de ganho. A última *inlet* é responsável pela entrada do valor da freqüência de oscilação do sinal modulador. Como *output* esta unidade envia o sinal resultante da síntese AM.




*Objeto [AM~]*

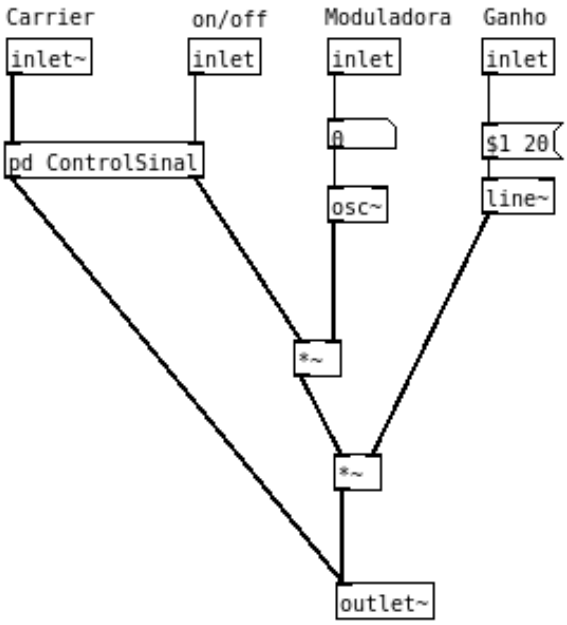
**RM**

Assim como o módulo AM este módulo representa uma técnica de síntese clássica que opera sobre uma modulação da amplitude de um sinal. Esta técnica se chama Modulação em Anel (*Ring Modulation)*. No entanto, diferentemente da síntese AM, na Modulação em Anel (RM) são multiplicados dois sinais *bipolares*. Logo, a amplitude da *carrier* é determinada inteiramente pelo sinal modulador (MIRANDA, 2002).

Como na síntese AM, quando a freqüência da moduladora está abaixo de 20Hz ouve-se o mesmo efeito *trêmolo*, e ao ultrapassar o sub-áudio ouve-se um timbre em decorrência do surgimento de parciais. No entanto, na síntese RM, o sinal da *carrier* se anula, e ouve-se apenas o som resultante das duas parciais correspondentes (ROADS, 1996).

Esta unidade foi baseada na criação do objeto [RM~]. Este objeto possui quatro *inlets*, sendo da direita para a esquerda, a primeira responsável pela entrada de um sinal de áudio (*carrier*), as

duas conseqüentes pelo controle de acionamento da unidade e ajuste do ganho, consecutivamente, e a última pela entrada do valor da freqüência de oscilação do sinal modulador. Como *output* a unidade envia um sinal de áudio resultante da síntese RM.




*Objeto [RM~]*

**FM**

O módulo FM é a última unidade das quais chamamos de *moduladores de sinal*. Este módulo corresponde à técnica de síntese clássica chamada Modulação de Freqüência (*Frequency Modulation*).

Vimos anteriormente duas técnicas de síntese que tinham a amplitude do sinal original modificada, ou modulada, a partir da multiplicação com outro sinal. Na Modulação de Freqüência (FM) o que ocorre é que um sinal tem sua freqüência modulada pela adição de outro sinal (ROADS, 1996). Neste tipo de modulação há três parâmetros envolvidos: a *carrier*, a moduladora e o *index* de modulação. Ao adicionar a moduladora ao sinal original tem-se como resultado uma variação de freqüência deste último. Esta variação é determinada pelo *index* de modulação que é representado pela amplitude da moduladora (ROADS, 1996).

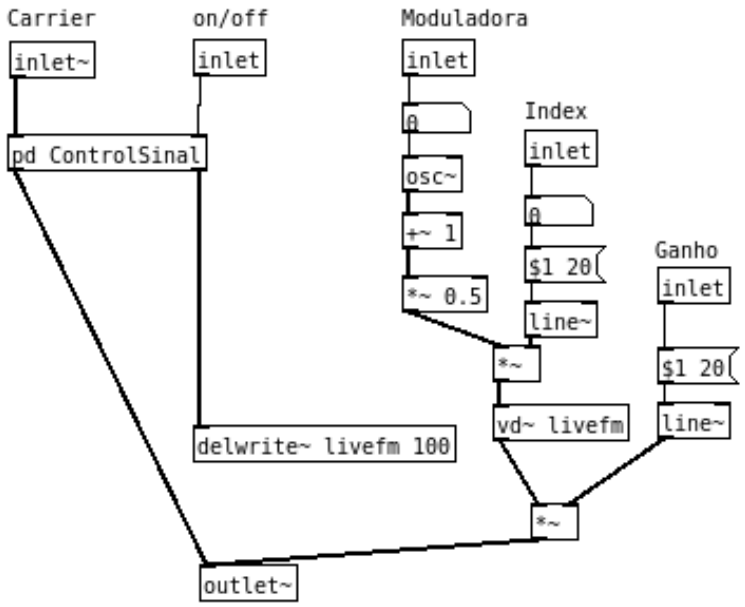
Se o valor da freqüência da moduladora estiver abaixo do limiar de audição(20Hz) ouviremos um efeito *vibrato* ou, dependendo da extensão do *index* de modulação, um efeito *glissando*. Para calcular a taxa de variação da freqüência basta somar e subtrair o valor da *carrier*

pelo *index* de modulação. Por exemplo, se tivermos uma *carrier* de 400Hz e um index de modulação de 100 a taxa de variação da freqüência será de 300Hz – 500Hz , um *glissando*.

Assim como nas modulações anteriormente vistas, quando a freqüência da moduladora ultrapassa 20Hz são adicionadas parciais ao sinal original. No entanto, na síntese FM, podem ser adicionadas mais do que duas parciais. O número de parciais é determinado pela quantidade do *index* de modulação. A adição das parciais pode se apresentar de forma harmônica ou inhamônica, dependendo da relação de freqüência entre a *carrier* e a moduladora.

A Modulação de Freqüência quando acima de 20Hz também conserva a freqüência fundamental, no entando, ao aumentar o *index* de modulação aumenta-se o número de parciais e suas amplitudes e, por este motivo, a *carrier* pode "ofuscar-se".

Esta unidade foi baseada no objeto [LiveFM~]. Este objeto tem como essência a concepção de síntese FM com *delay* variável, de Kreidler (KREIDLER, 2009). Este objeto possui cinco *inlets*, sendo da direita para esquerda: *carrier*, *on/off*, freqüência da moduladora, *index* de modulação e ganho. Como *output* o objeto [LiveFM~] envia um sinal de áudio resultante da síntese.

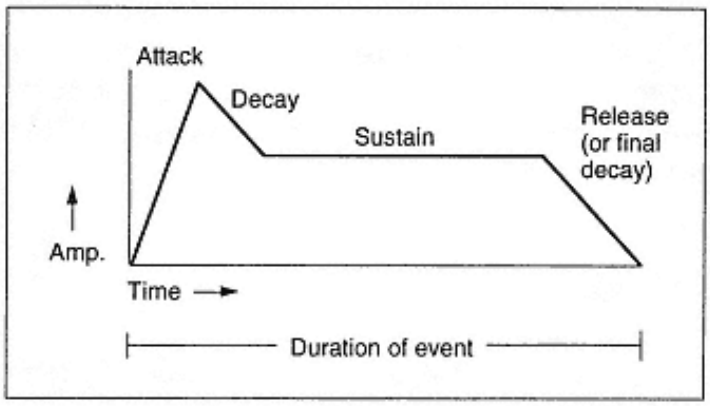



*Objeto [LiveFM~]*

**ADSR**

O ADSR é um dos módulos dos quais chamamos de *esculpidor subtrativo*. Demos este nome pelo motivo de que este módulo é capaz de processar um sinal original "cru", aparentemente

sem forma definida, e esculpi-lo em um "gesto instrumental". Tal propósito é possível pela manipulação de dois atributos do sinal sonoro: o tempo e a amplitude de energia. Podemos ilustrar esta relação por meio de um gráfico, no qual o eixo horizontal representa o tempo e o eixo vertical a amplitude do sinal:




*Envelope ADSR - figura retirada de ROADS (1996)*

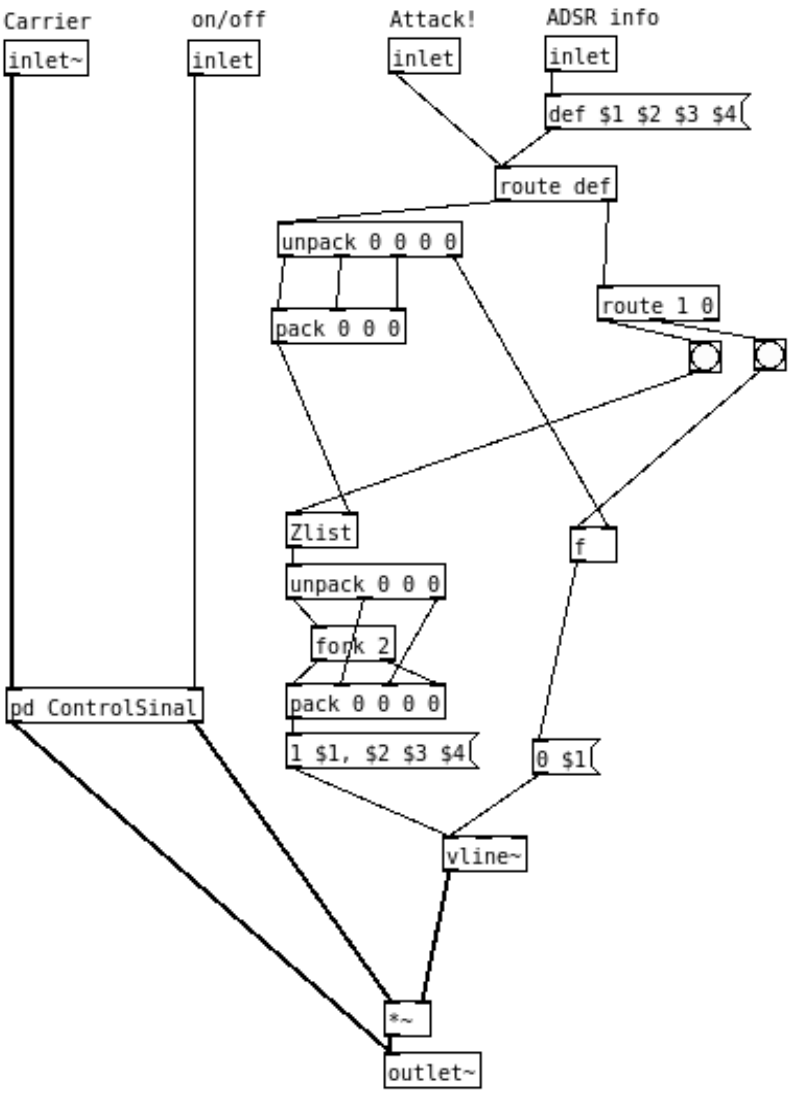
Este tipo de transformação é denominado de *envelope de amplitude* (ROADS, 1996). O envelope pode aparecer de várias maneiras de acordo com o controle da amplitude, no entanto, ao longo dos anos o modelo ADSR sustentou-se como o preferido (PUCKETTE, 2007). ADSR é um acrônimo para as palavras em inglês: *Attack*, *Decay*, *Sustain* e *Release*. O parâmetro *Attack* corresponde ao tempo gasto pelo sinal do seu ponto de energia nulo até o ponto máximo. *Decay* representa a primeira queda da amplitude, ou perda de energia, do sinal após o "golpe" inicial. O parâmetro *Sustain* equivale ao que se poderia chamar de o "corpo" do sinal, que permanece até sua queda final (*Final Decay)* ou, *Release*.

O *envelope de amplitude* corresponde a uma técnica de síntese inspirada na propriedade de dinâmica de intensidade dos instrumentos acústicos. Podemos exemplificar o envelope ADSR em dois instrumentos. Ao compararmos um oboé e um tímpano orquestral veremos que o parâmetro *Attack* no primeiro é muito mais atenuado que no segundo, enquanto o parâmetro *Sustain* a relação é inversa.

Esta unidade foi baseada na construção do objeto [ADSR~]. Este objeto tem como infra-estrutura o objeto [vline~], que pertence a biblioteca nativa do Pd. O [ADSR~] possui em sua concepção a definição dos valores do envelope ADSR e seu acionamento a partir de um gatilho. Este gatilho permite também o controle da duração entre S e R.

Este objeto possui quatro *inlets*. A primeira, da direita pra esquerda, recebe o sinal a ser

envelopado. As duas seguintes, uma pelo acionamento da unidade(*on/off)* e outra pelo acionamento de sua função. A última *inlet* é responsável por receber os valores de ADSR. Como *output* o objeto envia um sinal com os parâmetros de ADSR.




*Objeto [ADSR~]*

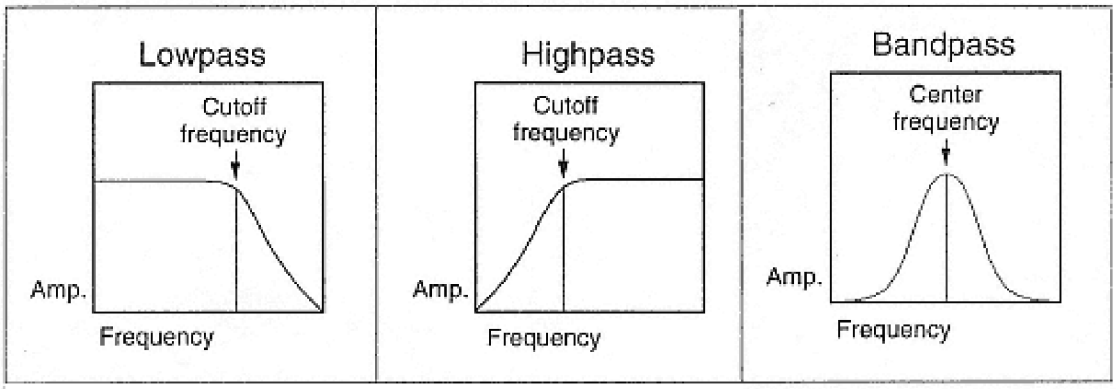
**FILTRO**

O módulo FILTRO corresponde à unidade de processamento que representa a técnica de síntese clássica chamada de Síntese Subtrativa. Este tipo de síntese trabalha de forma inversa à Síntese Aditiva. Ao invés de partir da sobreposição daquilo que poderíamos chamar de a menor

unidade no som – a onda senoidal – para construir sons complexos, a Síntese Subtrativa parte destes, geralmente o ruído branco, para "esculpir" o espectro sonoro (KREIDLER, 2009). Isto é possível por meio do uso de filtros de freqüência.

Um filtro de freqüências pode ser descrito como um mecanismo que efetua uma operação sobre um sinal atenuando ou intensificando regiões do espectro (ROADS, 1996). Temos basicamente três tipos de filtros: *Lowpass*, *Highpass* e *Bandpass*.

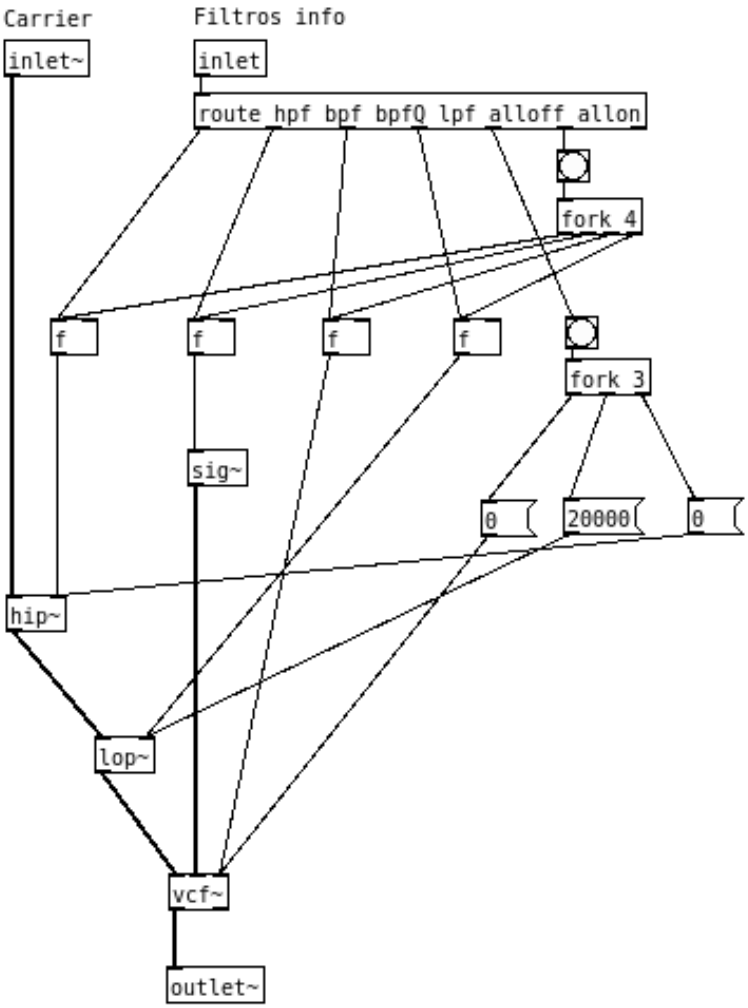
O filtro *Lowpass*, ou passa-baixa, permite a passagem das freqüências que estão abaixo de um limiar de corte e atenua as freqüências acima deste limiar. O filtro *Highpass*, ou passa-alta, opera de maneira inversa, corta as freqüências que estão abaixo do limiar de atenuação e intensifica as freqüências acima deste limiar.




*Filtros - figura adaptada de ROADS (1996)*

O filtro *Bandpass*, ou passa-banda, permite a passagem de uma faixa de freqüências acima e abaixo em relação a uma freqüência central. A extensão da faixa de freqüências permitida é controlada pelo coeficiente de ressonância *Q*.

Esta unidade foi baseada na construção do objeto [FILTRO~]. Este objeto possui duas *inlets*. A *inlet* esquerda recebe o sinal de áudio e a *inlet* direita recebe as informações das faixas de corte e *Q*. O núcleo deste objeto é composto pelos objetos [hip~], [lop~] e [vcf~], que são objetos nativos do Pd. Como output, [FILTRO~] envia um sinal de áudio resultante da síntese subtrativa. O sinal pode também receber ganho de amplitude por meio de um potenciômetro instalado no filtro.




*Objeto [FILTRO~]*

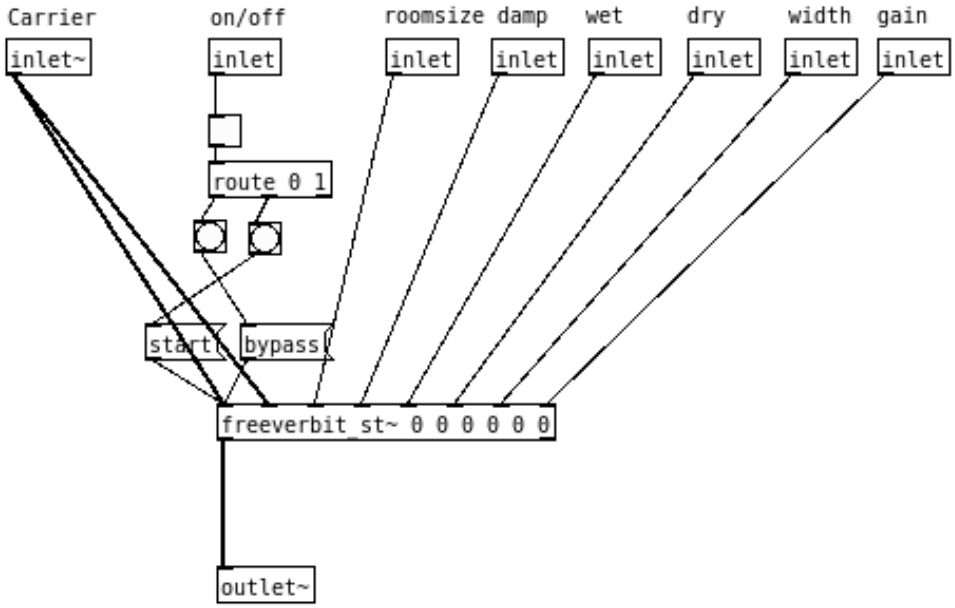
**REVERB**

Este módulo representa a emulação do fenômeno da reverberação. A reverberação é um efeito acústico que ocorre naturalmente em espaços físicos. O conceito básico é que toda fonte sonora possui um sinal originalmente gerado e vários outros que são decorrentes deste pela reflexão e absorção dos corpos físicos de um ambiente ideal (ROADS, 1996).

A Reverberação Artificial é possível por meio do uso de filtros, que simulam as absorções do sinal original, e pelo emparelhamento de linhas de *delay* que simulam suas reflexões no espaço físico (PUCKETTE, 2007).

A unidade REVERB foi baseada na construção do objeto [REVERB~]. Este objeto tem como núcleo o objeto [freeverbit_st~] da biblioteca ZOOexternals de Marcus Bittencourt, que trata-se de uma implementação do modelo de *reverb* de J.A. Moorer e M.R. Schroeder feita por Jezar at Dreampoint. O objeto [REVERB~] possui oito *inlets*, sendo que as duas primeiras, da esquerda para

a direita, são responsáveis pela entrada do sinal de áudio e pelo acionamento da unidade (*on/off).* As demais *inlets* correspondem aos parâmetros de reverberação: tamanho do espaço ideal (*roosize),* quantidade de absorção (*damp*), porção do sinal a ser processado *(wet)*, porção do sinal original não processado *(dry)*, extensão do *reverb* no campo estéreo *(width)* e ganho do sinal processado (*gain).* Como *output* este objeto envia um sinal de áudio resultante do processo de reverberação.

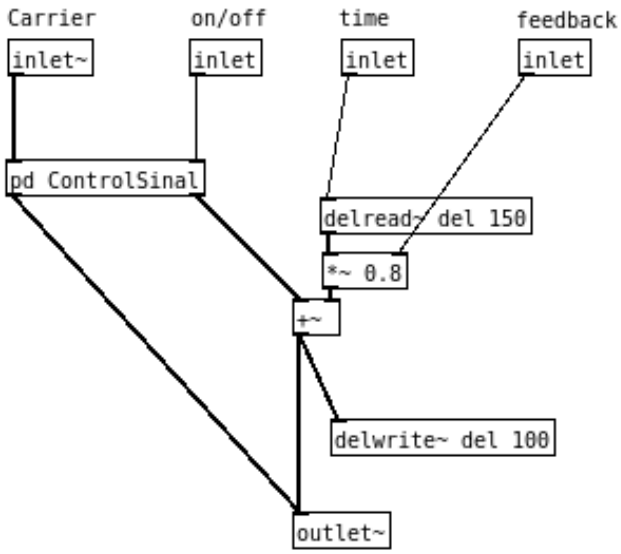



*Objeto [REVERB~]*

**DELAY**

Assim como o *reverb,* a unidade DELAY compreende um efeito sonoro. O *delay* é um efeito no qual o som original é reprocessado com um atraso em relação ao seu precedente. Geralmente os sinais que estão sobre a ação do *delay* são reprocessados com um ganho menor, resultando em um efeito *eco* [2].

Esta unidade foi baseada no objeto [DELAY~], que foi construída com os objetos [delread~] e [delwrite~], que pertencem à biblioteca nativa do Pd. O objeto [DELAY~] possui quatro inlets, que possui como argumentos, da esquerda para a direita: entrada para o sinal original, *on/off* da unidade e os parâmetros do *delay*, tempo de atraso entre os sinais *(time)* e ganho inicial dos sinais em efeito *delay* (feedback).




*Objeto [DELAY~]*

**TECLADO, GRAVADOR e DAC**

Após o processo de síntese sonora o sintetizador necessita de um meio pelo qual este resultado seja acionado e enviado para um *output* analógico. Esta função é representada pelos controladores. No caso, a propósito de teste, escolhemos uma adaptação de um teclado virtual (unidade TECLADO) criado por Andres Ferrari [3].

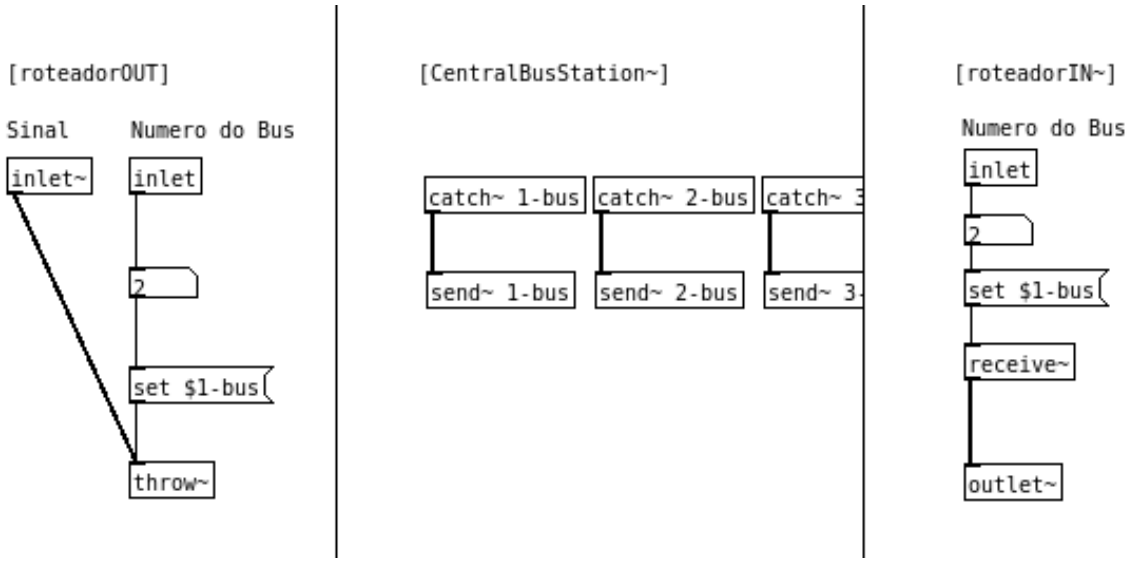
O caminho do sinal digital após seu acionamento pelo controlador pode se dirigir tanto para o *output* analógico, representado pela unidade DAC, quanto para a unidade GRAVADOR, que faz o armazenamento do áudio digital. O módulo DAC foi criado com o objeto [dac~], nativo do Pd e GRAVADOR foi criado com os objetos [TapeRecorder~] e [X_sfplayer~] da biblioteca ZOOexternals de Marcus Bittencourt.

**Ligando os cabos: Sistema de *Buses* internos e *Presets***

Poder-se-ia imaginar que, a partir de uma construção baseada nos objetos [send~] e [receive~] estaria resolvido a interconexão entre os módulos do sintetizador. No entanto, a sintaxe destes objetos impedia tal proposição. O objeto [send~] não possui método para se criar conexões dinâmicas. Se usássemos estes objetos teríamos inúmeros [receive~] dentro de cada módulo correspondentes ao número das unidades do sintetizador e todas elas estariam recebendo um sinal, mesmo não participando do circuito de síntese. O mesmo acontece com outra dupla de objetos do

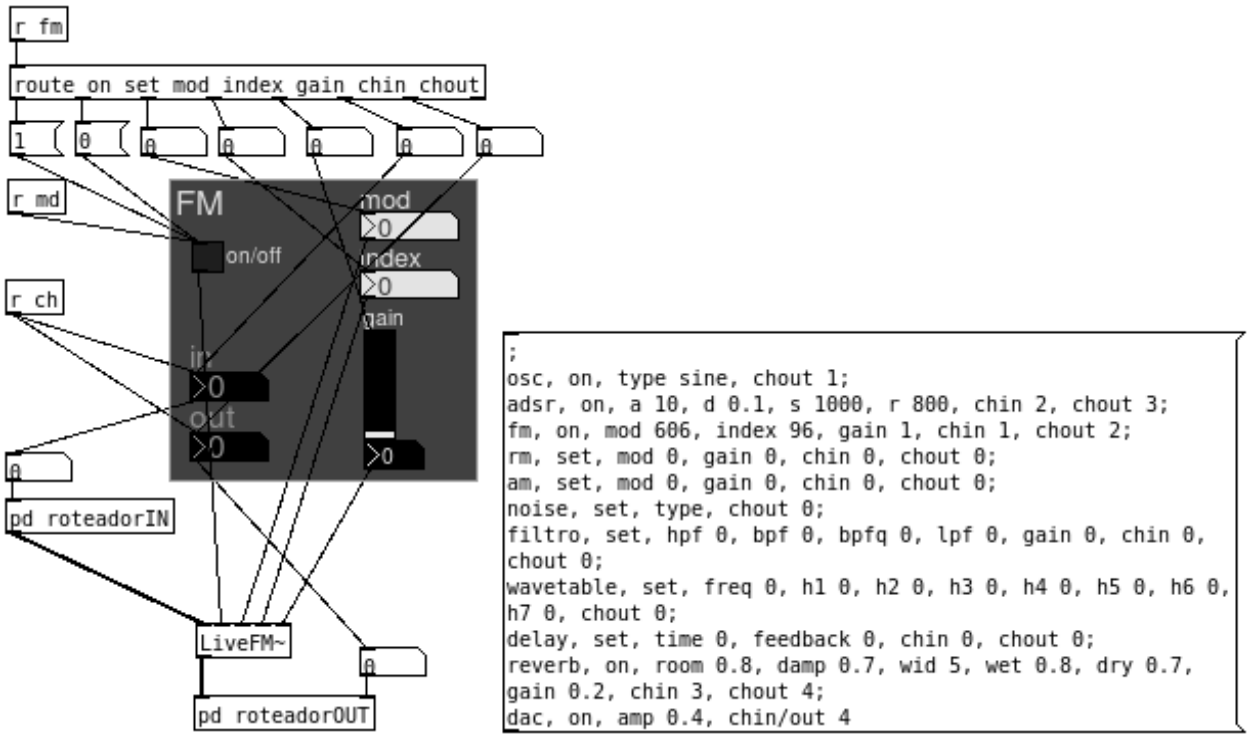
Pd que fazem conexões sem fio: [throw~] e [catch~]. Neste caso, [catch~] não suporta conexões dinâmicas pois recebe um caminho de sinal fixo.

Logo, buscou-se utilizar os métodos que seriam eficazes a partir destes quatro objetos. Como resultado chegou-se à confecção de um sistema de *buses* internos. Este sistema aliou as possibilidades de definição do envio de sinal de [throw~] com a flexibilidade de [receive~] por meio de uma central de redefinição de métodos. Este sistema é compreendido pelos objetos [roteadorOUT~] e [roteadorIN~] instanciados em cada unidade de síntese, e [CentralBusStation~] alocado em uma unidade a parte chamada CENTRAL.



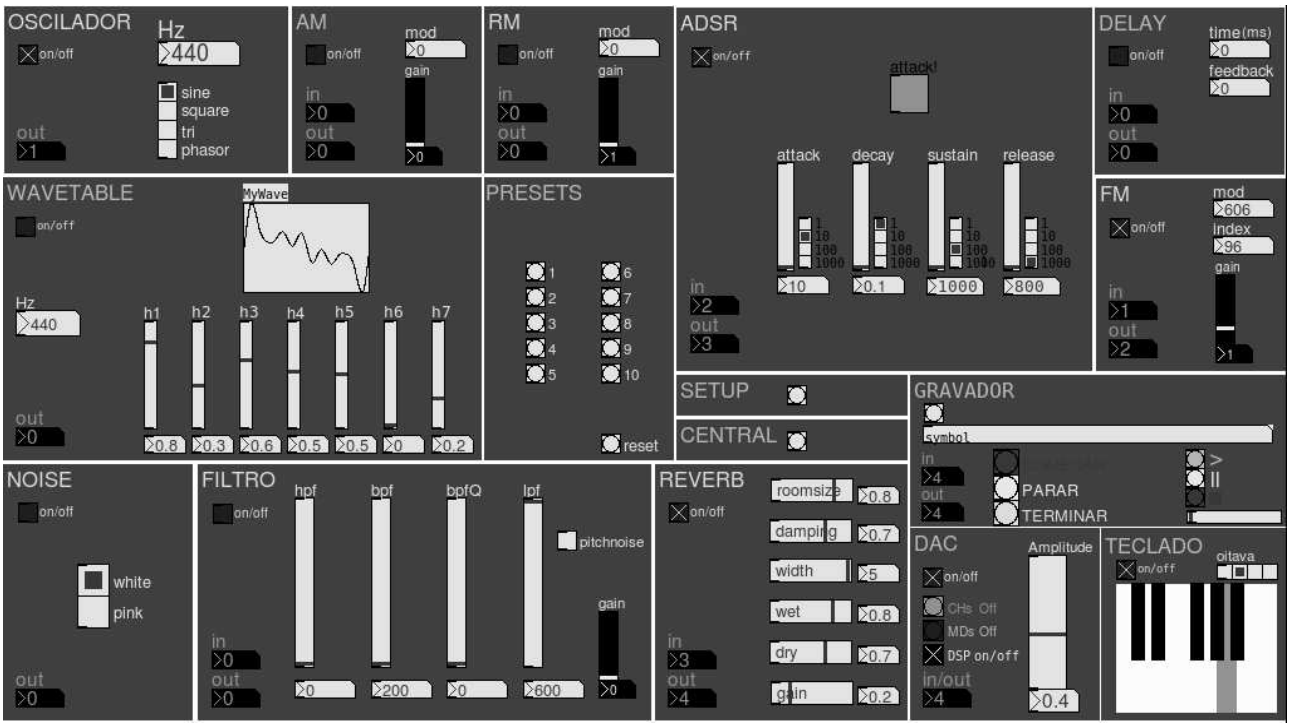

*Sistema de buses internos*

A partir da concepção do sistema de *buses* internos foi possível a pré-configuração de *setups* do circuito de processamento. Isto se deu por meio do uso do selecionador [route] e de um *send* em forma de mensagem instantânea. Os módulos referentes são SETUP e PRESETS.




*Sistema de setup e presets*

Enfim, após termos construído cada módulo com sua respectiva funcionalidade optamos em equiparar a aparência do sintetizador modular digital ao analógico a fim de buscar um modo mais intuitivo em se trabalhar com o processo de síntese sonora. Tal propósito foi possível a partir do uso de GUIs próprios da linguagem, com destaque especial para o método *graph on parent*. Este método pertence a GUI *canvas* e permite criar, a partir de uma *abstract*, uma unidade de programação individual (um objeto) que deixa transparecer apenas os elementos de interface gráfica. Por ser um objeto, as mesmas características de manipulação também são iguais, logo, pode-se duplicar a unidade ou aloca-la no espaço de trabalho do Pd, criando uma grande interface gráfica.




*Sintetizador modular virtual em Pure Data*

## 6 – Considerações Finais

Primeiramente, os resultados obtidos neste projeto de iniciação científica identificam-se pela própria confecção do sintetizador modular digital. Além de proporcionar o aprendizado de técnicas de síntese e de programação em Pure data, este projeto permitiu o experimento de possibilidades sonoras por meio do instrumento virtual muito próximas ao seu modelo analógico. Soma-se a isto a possibilidade futura de se adaptar ao sintetizador virtual a uma interface física via MIDI, como um teclado, por exemplo.

Espera-se também que os frutos deste trabalho possam instruir aqueles que estejam interessados em se enveredar pelos caminhos da Computação Musical. Para tal propósito o presente trabalho será disponibilizado no endereço eletrônico do Laboratório de Pesquisa e Produção Sonora (LAPPSO) do Departamento de Música da Universidade Estadual de Maringá (www.dmu.uem.br/lappso).

## 7 – Notas

[1] verbete “Programação procedural” da enciclopédia virtual Wikipedia, em http://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_procedural (acessado em 30 de agosto de 2010).

[2] em http://en.flossmanuals.net/PureData (acessado em 30 de agosto de 2010).

[3] em http://puredata.info/ (acessado em 20 de agosto de 2010).

### 7.1 Links importantes

Home page oficial do software Pure Data
http://puredata.info/

Site com informações sobre sintetizadores modulares
http://www.synthesizers.com/

Pagina oficial do Moog Synthesizer
http://www.moogarchives.com/

### 7.2 Referências Bibliográficas

CHADABE, Joel. *Electric Sound: The Past and Promise of Electronic Music*. Upper Saddle River, NJ: Prentice-Hall, 1997.

EIMERT, Herbert. *"What is Electronic Music?" die Reihe*, I (versão inglesa), Theodor Presser Londres, 1958.

PUCKETTE, Miller Smith. *The Theory and Technique of Electronic Music*. World Scientific Press, Singapore, 2007.

ROADS, Curtis. *The Computer Music Tutorial*. The MIT Press, Cambridge, MA, 1996.

MIRANDA, Eduardo Reck. *Computer Sound Design: Synthesis techniques and programming*, 2nd edition. Focal Press, Oxford, 2002.

KREIDLER, Johannes. *Loadbang: Programming Electronic Music in Pd*, versão inglesa, tradução: Mark Barden, 2009, Livro virtual disponível em http://www.pd-tutorial.com/ (acessado em 30 agosto de 2010).

PORRES, Alexandre. *Apostila do Curso de Computação Musical*, 2009, Tutorial de Pure Data disponível em http://sites.google.com/site/porres/home (acessado em 30 de agosto de 2010).