FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

MESTRADO EM MULTIMÉDIA

PLATAFORMA MODULAR PARA A PROTOTIPAGEM EM SISTEMAS MULTIMÉDIA INTERACTIVOS

Rui Miguel Silva Sampaio Dias

Dissertação submetida para satisfação parcial do grau de Mestre em Multimédia

Dissertação realizada sob a orientação do Professor Doutor Carlos Guedes, Professor Coordenador do departamento de Música da Escola Superior de Música e das Artes do Espectáculo do Porto e sob a co-orientação do Professor Doutor Eurico Carrapatoso, Professor Auxiliar do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Porto 2009

Agradecimentos

Ao Professor Doutor Carlos Guedes, caro professor, colega e amigo, com quem continuo a aprender tanto hoje como quando o conheci em 2002, como seu aluno de composição electroacústica na Escola Superior de Música e das Artes do Espectáculo do Porto, e pelo enorme apoio e confiança que sempre me transmitiu.

Ao Professor Doutor Eurico Carrapatoso, pela dedicação e disponibilidade, e pela mente aberta nesta área tão heterogénea.

Ao Professor Doutor Pedro Rebelo, do *Sonic Arts Research Center* de Belfast, pelas sugestões pertinentes.

Aos meus superiores na Escola Superior de Artes Aplicadas de Castelo Branco, onde lecciono desde 2005, o Professor Fernando Raposo, director, e o Professor José Raimundo, coordenador do departamento de música, pelo apoio e pela completa autonomia que me deram na gestão do meu tempo.

Aos participantes nos testes ao protótipo:

Aos meus alunos do curso de Multimédia da Escola Superior de Artes e Design de Matosinhos que me fizeram estar certo da utilidade do meu projecto, e particularmente ao Guilherme Gomes, Leonardo Guedes, Manuel Fardilha e Ricardo Gomes, autores da instalação *Polligital*.

Aos meus caros amigos José Tavares, José Marques e Cláudio Alves da Associação ACERT, em Tondela, pela disponibilidade que tiveram e pelo afincado interesse nestes assuntos das novas tecnologias.

À minha colega Ana Leite por ter tido a boa fé de querer usar os meus objectos, ainda em fase tão precoce.

Aos meus pais por todo o apoio que me têm dado nos últimos 34 anos e 11 meses. Ao meu irmão Zé e às minhas irmãs Lisete e Susana, por estarem lá.

Dedicado à Isabel e aos meus filhos Nuno e Sofia.

ÍNDICE

1. INTRODUÇÃO

Esta dissertação insere-se no contexto da programação para novos média e sistemas multimédia interactivos.

Serão ilustradas e contextualizadas as áreas de acção implicadas, identificados os problemas que constituíram os motivos e questões de base para esta dissertação, e será apresentada a implementação de um protótipo para uma possível solução para os problemas, recorrendo a um dos pacotes de software mais utilizados para a programação deste tipo de sistemas.

A solução que será apresentada, funcional e resultante da pesquisa levada a cabo para a presente dissertação, poderá trazer benefícios directos na implementação de projectos em áreas como sistemas digitais interactivos, *live performance*, síntese e processamento de som em tempo real, composição, composição em tempo real, *video jamming*, comunicação midi, improvisação, dispositivos de interface (controladores e sensores), comunicação em rede, arte generativa e programação em tempo real.

1.1.Motivação

1.1.1. Do geral...

Da minha experiência profissional, artística e pedagógica, revelou-se a necessidade de desenvolver uma plataforma flexível, coerente e acessível para a programação em sistemas multimédia interactivos, que permitisse, por um lado, uma maior fluência e rapidez no desenvolvimento de projectos nesta área para programadores experientes, e por outro lado uma utilização simplificada e mais intuitiva para utilizadores pouco experientes, e com poucos conhecimentos técnicos.

Esta necessidade prende-se especialmente com o facto deste ser um campo iminentemente multidisciplinar, onde se cruzam várias áreas de conhecimento, o que torna extremamente difícil o domínio de todos os aspectos envolvidos. Para além disso, desenvolver um projecto

neste campo é um exercício de criatividade e originalidade, e está directamente relacionado com o conhecimento dos recursos técnicos disponíveis.

Desenvolver um projecto num determinado domínio passa por escolher e utilizar a(s) ferramenta(s) apropriada(s) para esse domínio. Para realizar uma animação 3D para o cinema, por exemplo, é necessário utilizar um programa 3D, desenvolvido e completamente optimizado para a criação de gráficos 3D, tal como, para gravar um grupo musical num estúdio, por exemplo, é necessário usar um sequenciador ou outro programa de gravação próprio, desenvolvido e optimizado para esse efeito.

Pelo contrário, desenvolver um projecto no campo dos sistemas multimédia interactivos requer a utilização de ferramentas não específicas, que permitam lidar com vários tipos de recursos, uma vez que cada projecto pode ter características e componentes completamente diferentes e por isso se torna impossível situar num determinado domínio específico.

1.1.2. ... ao particular

O ambiente de programação Max, da empresa *Cycling'74* (www.cycling74.com), tem sido um dos mais relevantes na área de sistemas digitais interactivos e performance em tempo real, e conta com uma grande comunidade de utilizadores das mais variadas áreas.

Tratando-se de um ambiente gráfico, o ambiente Max proporciona de raiz uma interface intuitiva e acessível, relativamente a linguagens de programação por código, cuja sintaxe e leitura obrigam geralmente a uma curva de aprendizagem mais lenta.

Com a capacidade de ser expandido com inúmeras bibliotecas de objectos desenvolvidas por vários utilizadores e instituições, o Max disponibiliza recursos para processamento de som, vídeo, MIDI, interfaces gráficas, OSC, OpenGL, dispositivos de interface, microcontroladores, rede, etc.

O protótipo concebido no âmbito desta dissertação foi desenvolvido em Max, e pretende ser uma contribuição para o melhoramento e simplificação do desenvolvimento de projectos neste ambiente, no contexto dos sistemas multimédia interactivos.

1.1.3. Dos problemas...

Apesar de ser mais intuitivo e acessível, o Max foi inicialmente pensado como uma forma gráfica e modular de programar em C, e, como tal, concebido para ser o mais aberto possível e não limitativo para o utilizador. Para isto, as funções básicas disponibilizadas são, tal como em qualquer linguagem de programação, um conjunto de primitivas, e por si só não realizam operações completas. Isto é, são utilizadas para poderem ser combinadas pelo programador, de forma a construir a sua própria rotina ou função específica.

Esta é uma característica importante e uma das razões para que o Max rapidamente se tenha tornado uma referência e continuado a crescer ao fim de 23 anos de existência.

No entanto, além dessas primitivas, a estrutura de um programa é constituída geralmente por um conjunto de pequenos procedimentos e subrotinas que são mais comuns, e utilizadas com muita frequência, em várias aplicações distintas.

Assim, e apesar das vantagens inerentes a uma concepção aberta e abrangente como a do Max, a programação **revela-se frequentemente extenuante, pela repetição a que obriga no trabalho com procedimentos relativamente comuns**.

Este facto prende-se, a meu ver, com dois aspectos que, à medida que o nível de utilização e a quantidade e complexidade dos recursos implicados foram aumentando, se vêm tornado problemáticas:

1º - interface de utilizador

Sendo um ambiente gráfico, um programa (habitualmente designado por *patch*) é definido por ligações (*patchcords*) entre os vários tipos de elementos que o constituem.

No entanto, podemos distinguir dois tipos de ligações, ainda que visualmente possam ser exactamente iguais:

- ligações estruturais, que definem a estrutura do algoritmo
- ligações locais, que definem parâmetros e mensagens de controlo para os objectos.

Um *patch* tem, geralmente, muitas mais ligações locais do que estruturais. Num *patch* de média ou grande complexidade este aspecto pode tornar o patch muito complicado visualmente e muito difícil de perceber e fazer alterações.

Na **Fig. 1-1** podemos ver um pequeno *patch* de processamento de vídeo. Em a) vemos o patch normal, com todas as ligações e mensagens, e podemos contar dezoito elementos. No entanto, os elementos estruturais do algoritmo, que constituem a estrutura *dataflow*, podem ser reduzidos a apenas 5, como se pode ver em b). Todos os outros treze são parâmetros e ordens de comando para definir e controlar os cinco elementos estruturais (ver c)).

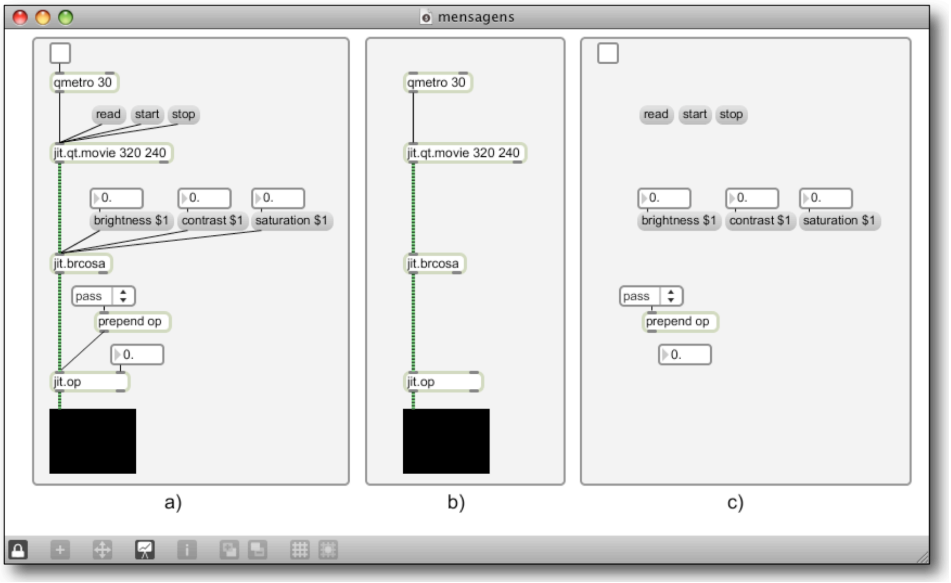


Fig. 1-1 Ligações estruturais e ligações locais

Não é por isso difícil chegar a algo assim que, como se pode ver, torna muito difícil o debugging e manutenção do código do programa:

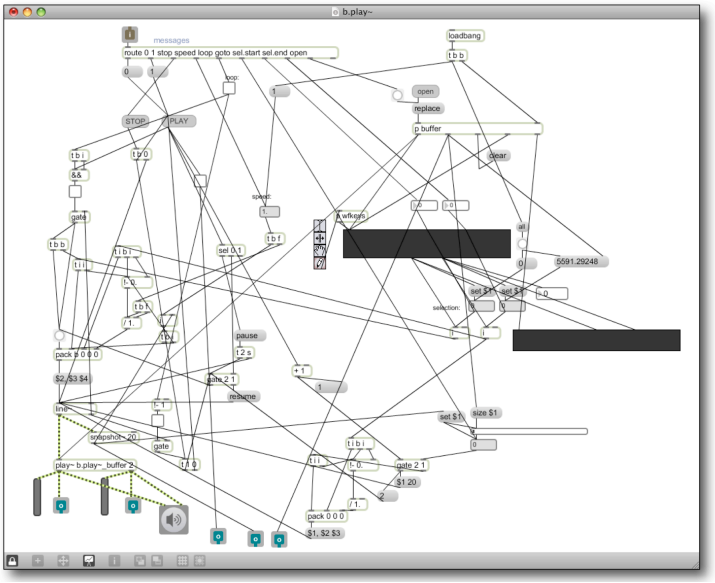


Fig. 1-2 Um patch facilmente pode ter este aspecto

O Max tem implementados alguns recursos para fazer face a este problema. Os objectos e *patchcords* podem ser escondidos, secções do patch podem ser encapsuladas, e, a partir da versão 5, o modo de apresentação (ver pág. 27) veio facilitar imenso a organização do patch, ao criar uma visualização independente para o layout de controlo.

No entanto, o problema das ligações locais permanece, uma vez que é preciso colocá-las e organizá-las, todas as vezes que as queremos usar.

Nota: a versão 5 introduziu ainda vários melhoramentos a nível de interface e facilidade de utilização, como por exemplo atalhos de teclado para criar os vários objectos, múltiplos níveis de *undo*, etc. que já há bastante tempo vinham sendo pedidas na comunidade de utilizadores.
Além disso, desde a versão 4.x foi adicionada a capacidade de usar **java** e **javascript** dentro do Max.
Uma adição especialmente útil é o *Max Toolbox*, de *Nathanaël Lécaudé*, que permite, através de atalhos de teclado, fazer automaticamente as ligações entre os objectos, bem como distribui-los horizontal e verticalmente.

2º - modularidade

Este aspecto está relacionado com toda a concepção de base deste ambiente. Apesar de ser um ambiente de programação e não uma linguagem, o Max está muito próximo ainda do nível de uma linguagem de programação de baixo nível, no que diz respeito à universalidade das funções que disponibiliza. Ou seja, a maior parte dos objectos não são pensados para fins específicos, mas sim de uma forma genérica.

Para inverter as cores de uma imagem com os objectos Jitter, por exemplo, é necessário usar o objecto *jit.op*, com o operador *absdiff* e um valor de 1 (ver **Fig. 1-3**). Se, para um utilizador experiente isto pode não ser grande problema, para utilizadores médios e principiantes, isto pode ser muito difícil de descobrir. Um objecto com o nome de *jit.invert* ou *jit.negative*, seria muito mais prático e intuitivo. Contudo, isto iria contra todas a ideia de base da concepção do Max (e Jitter, neste caso), porque este objecto ficaria associado a uma função específica – inverter as cores – enquanto

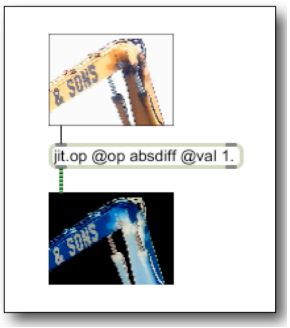


Fig. 1-3 Inversão de cor com o objecto *jit.op*

que a combinação *jit.op-absdiff-1* pode ser usada para muitas outras finalidades, mesmo sendo exactamente o mesmo processo.

Não se pode dizer que este aspecto seja um problema, mas sim uma característica, coerente com o facto de que pode ser utilizado em áreas completamente distintas e por isso não deve ser específico. É com as bibliotecas externas que podemos encontrar funções mais especializadas, com objectos concebidos para determinadas finalidades específicas.

Contudo, desenvolver um projecto multimédia interactivo pode ser muito complicado porque obriga a conhecer e saber implementar todos os processos envolvidos, ou a conhecer e gerir vários tipos de bibliotecas no mesmo patch, o que, em certos casos, pode ser ainda mais complicado.

1.1.4. ... à solução

Surge então a **necessidade de construir uma plataforma de mais alto nível, para a programação em sistemas multimédia interactivos**, que disponibilize, de forma coerente e acessível, recursos frequentemente utilizados e permitam simplificar a programação em Max.

Na construção de uma determinada aplicação há funções e procedimentos que são específicos e concebidos apenas para a criação dessa aplicação, e que não seria possível prever ou não faria sentido numa linguagem de programação. No entanto, a estrutura de um programa é constituída geralmente por um conjunto de pequenos (ou não) procedimentos e subrotinas que são utilizadas com muita frequência, em várias aplicações distintas.

Para um programador menos experiente, estas subrotinas são frequentemente obstáculos intransponíveis, que requerem uma aprendizagem mais profunda da sintaxe e convenções do programa, e que por vezes podem ser bastante complexas de implementar. Ao facilitar o acesso a certos tipos de recursos, e assim centrar os alunos na concepção da aplicação a que se destina, torna estes recursos não só mais fáceis de ensinar como mais apelativos para os alunos, uma vez que conseguem ver mais rapidamente os resultados, e assim também perceber de forma mais clara o funcionamento destas.

É na identificação e implementação destas subrotinas, que muitas vezes são quase aplicações por si só, que se centra este trabalho, por ser pertinente e útil para reduzir o tempo necessário para a programação de aplicações em Max.

Uma solução integrada e coerente que aborde este problema irá tornar-se uma ferramenta útil, quer em situações profissionais para utilizadores mais experientes, para que não tenham de estar constantemente a "inventar a roda", quer, num plano pedagógico, para ajudar a leccionação e utilização do software em questão.

Essa solução deve também abordar o problema da interface, minimizando ou eliminando, sempre que possível, o problema das ligações locais.

Assim, a solução aqui proposta é a criação de uma biblioteca de abstracções com interfaces gráficas que foquem vários procedimentos de utilização recorrente, e disponibilizem ao utilizador vários recursos multimédia optimizados para uma utilização mais rápida e simplificada, especialmente direccionada para a prototipagem deste tipo de sistemas.

1.2.Objectivos

O objectivo principal deste trabalho é o desenvolvimento de uma solução que permita simplificar e facilitar a programação em Max/MSP/Jitter.

Esta solução é uma biblioteca de abstracções com interface gráfica, intitulada ***Max Building Blocks***.

Esta biblioteca criará uma plataforma de mais alto nível especialmente pensada para a programação em sistemas multimédia interactivos, que, no entanto, coabitem perfeitamente com os objectos Max, e que não comprometam a performance nem a versatilidade dos recursos originais.

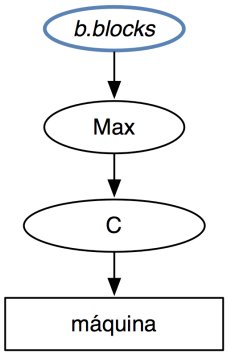


Fig. 1-4 Esquema conceptual da plataforma

As abstracções estão agrupadas por tópicos, em colecções relativas à análise e processamento de imagem, análise e processamento de áudio, comunicação e funções globais.

Uma vez que as todas as abstracções têm a sua própria interface, o problema das ligações locais fica resolvido, para grande parte das utilizações dos módulos.

A Fig. 1-5 mostra o mesmo patch da Fig. 1-1 (pág. 7), implementado com os *b.blocks*.

Os comandos mais comuns estão disponíveis e visíveis. Desta forma o utilizador fica imediatamente a saber quais são os parâmetros principais, e não tem de criar caixas de mensagem nem fazer nenhuma ligação para os utilizar.

Fig. 1-5 exemplo de patch de processamento de vídeo com módulos b.blocks

2. ENQUADRAMENTO

A utilização de tecnologias digitais interactivas tem vindo a aumentar visivelmente em várias áreas distintas, possibilitando novas abordagens e paradigmas para a criação de novas soluções para contextos artísticos, aplicações comerciais, aplicações lúdicas e educativas e também para contextos académicos. Abaixo serão descritos e ilustrados estes tópicos.

2.1. Aplicações

2.1.1. Sistemas multimédia interactivos em contextos artísticos e culturais

A existência de projectos interdisciplinares nas artes tem já antecedentes no séc. XIX, nomeadamente com o conceito de obra de arte total (*Gesamtkunstwerk*), idealizado pelo compositor Richard Wagner (in Packer & Jordan, 2002), e no séc. XX, com os movimentos artísticos da primeira metade do século, e de uma forma mais concreta e sistematizada na década de 60 com o *Happening* e movimento *Fluxus*.

Em 1965, para o projecto multidisciplinar *Variations V*, do compositor John Cage, com Gordon Mumma e David Tudor, criado para a companhia de dança do coreógrafo Merce Cunningham, foi criado por um sistema que utilizava vários sensores fotoeléctricos e antenas colocadas no palco, para detectar a passagem e proximidade dos bailarinos, para gerar ou modificar sons (Winkler, 1995).

Fig. 2-1 *Variations V* (1965)

O recurso às tecnologias digitais permitiram o desenvolvimento de toda uma nova categoria de projectos, como a música com electrónica em tempo real e o *vídeo jamming*, entre muitos outros.

As aplicações podem ser tão simples como por exemplo o controlo de dispositivos de iluminação pela detecção de som, até situações de enorme complexidade como por exemplo

a *VirtOpera* (art+com, 2002) uma ópera do compositor André Werner, na qual todo o cenário e os figurinos são virtuais e gerados em tempo real, com um sistema interactivo desenvolvido pela empresa alemã *art+com* (www.artcom.de).

Fig. 2-2 Cenas da *VirtOpera*, de *André Werner*, onde se podem ver o cenário e figurinos virtuais

2.1.2. Sistemas multimédia interactivos para aplicações comerciais

O recurso a este tipo de sistemas em aplicações comerciais é já uma realidade que tem vindo a crescer significativamente. Apesar de serem de uma forma geral ainda relativamente simples, facilmente se poderá verificar que o aspecto da interactividade com o utilizador ou o público se pode revelar uma grande mais valia, nomeadamente em áreas como o marketing e a publicidade.

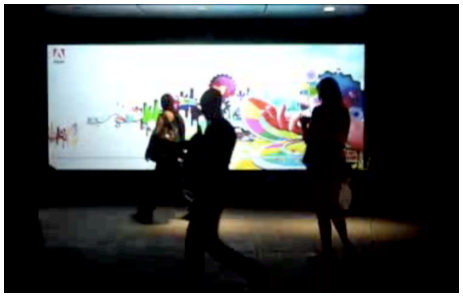


Fig. 2-3 Adobe CS3: instalação multimédia interactiva

Num plano internacional podemos ver como exemplos a campanha de lançamento do pacote de software *Creative Suite* 3 da empresa *Adobe*, em que foi instalado um ecrã de projecção numa vitrina da loja *Virgin Megastore* na *Union Square* em Nova York, que mostrava gráficos e imagens interactivas, que surgiam de acordo com os movimentos das pessoas que passavam na rua.

Em Portugal destacam-se as empresas **Ydreams** (ydreams.com), de Lisboa, e **Edigma** (edigma.com), de Braga. Ambas desenvolvem produtos e oferecem serviços inovadores com recurso às novas tecnologias de interacção, realidade virtual, realidade aumentada, comunicação, interfaces, entre outros, com aplicações em marketing, publicidade, educação, divulgação e comunicação.

2.1.3. Sistemas multimédia interactivos em aplicações lúdicas e educativas

A aplicação deste tipo de sistemas na indústria dos jogos para computadores e consolas será sem dúvida um dos mais significativos e de maior visibilidade no mercado, uma vez que o desenvolvimento de jogos e dispositivos de interface têm implicações criativas e técnicas muito semelhantes à criação de sistemas digitais interactivos. Nomeadamente, o desenvolvimento de novas soluções de interface homem-máquina, ponto central em muitas das aplicações interactivas desenvolvidas todas as áreas, sempre foi pratica corrente no desenvolvimento de jogos.

Como exemplos de modelos de interacção recentes na indústria dos jogos temos o sistema *Eyetoy* na consola Playstation da Sony e os controladores *Wiimote* e *Nunchuck* da Nintendo.

No entanto, uma vez que se trata de aplicações para um grande público, torna impossível o desenvolvimento de situações específicas e personalizadas, deixando assim de lado um dos aspectos mais interessantes das potencialidades dos sistemas interactivos.

2.1.4. Sistemas multimédia interactivos em contextos académicos

À semelhança do que se pode verificar num âmbito internacional, assistiu-se nos últimos anos ao aparecimento, em vários cursos do ensino superior público e privado, de disciplinas que abordam conteúdos específicos na área dos sistemas multimédia interactivos.

São exemplos disso as seguintes instituições de ensino:

- Faculdade de Engenharia da Universidade do Porto
- Escola Superior de Música e das Artes do Espectáculo do Instituto Politécnico do Porto
- Escola Superior de Artes e Design de Matosinhos
- Escola Superior de Artes Aplicadas de Castelo Branco
- Universidade da Beira Interior
- Escola de Artes da Universidade Católica, Núcleo Regional Norte

A inclusão destes conteúdos mostra claramente a relevância destes, junto do panorama profissional e artístico de várias áreas de actividade profissional actuais. Também o facto de se tratarem de conteúdos frequentemente interdisciplinares e transversais a várias áreas de conhecimento e competências técnicas, faz com que façam sentido em várias áreas de ensino, como as artes da imagem, música, tecnologias informáticas, electrónica, teatro, luz e som, produção, etc.

2.2. O ambiente de programação Max

O ambiente de programação Max foi criado em 1986 por Miller Puckette no *Institute de Recherche et Coordination Acoustique/Musique* (IRCAM), em Paris, como ferramenta de criação de música interactiva, possibilitando o controlo total de comunicação com o protocolo MIDI (*Musical Instruments Digital Interface*).

> *"Max is a graphical music programming environment for people who have hit the limits of the usual sequencer and voicing programs for MIDI equipment."*
>
> (Puckette, 1988)

Em 1991 tornou-se um produto comercial, distribuído pela companhia *Opcode Systems*, com desenvolvimento de Puckette e David Zicarelli, e desde 2000 passou a ser desenvolvido e comercializado pela empresa *Cycling '74*, em San Francisco, criada em 1997 por Zicarelli. Desde então foram também criadas as expansões ou bibliotecas *MSP* (*Max Signal Processing*) para processamento de áudio e *Jitter* para processamento de matrizes, optimizado para a criação e tratamento de vídeo, gráficos 2D e 3D.

O nome do programa é uma homenagem a *Max Mathews*, investigador nos laboratórios Bell e pioneiro da música por computador.

Baseado na linguagem de programação *C*, o ambiente Max é uma linguagem *dataflow*[1], e consiste numa interface gráfica, que permite criar aplicações complexas ligando pequenas "caixas" que desempenham funções específicas. Grande parte dessas funções são primitivas, semelhantes àquelas encontradas em todas as linguagens de programação, como operações matemáticas, objectos que implementam estruturas de repetição, estruturas de selecção, etc. e outras menos comuns, adaptadas a toda a lógica e particularidades específicas do Max. Essas pequenas caixas são combinadas de forma que um programa em Max (*patch* ou *patcher*) é muito semelhante a um fluxograma, tornando mais fácil quer a concepção de um programa, quer o seu *debugging*.

[1] *Dataflow* é uma arquitectura de software que tem por base o paradigma o fluxo dos dados e da actualização automática de todas as variáveis, sempre que alguma outra muda.

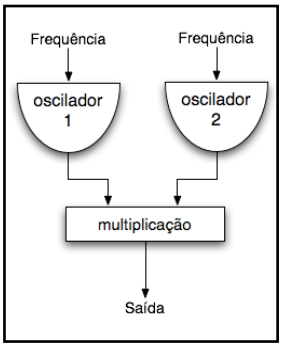


Fig. 2-4 Diagrama de fluxo de um circuito de modulação em anel

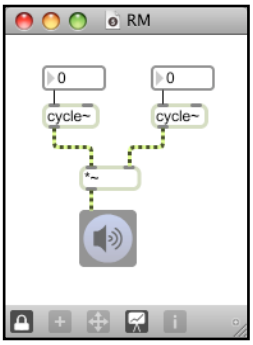


Fig. 2-5 O mesmo circuito implementado em Max

O Max é uma ferramenta de programação de alto nível, em que os programas são "escritos" com objectos gráficos em vez de texto (Zicarelli & Taylor, 2006).

A inclusão de várias bibliotecas de expansão torna o Max numa ferramenta poderosa e intuitiva, uma vez que o funcionamento integrado e coerente entre os vários tipos de objectos numa mesma interface vem simplificar e potenciar a criação para contextos multimédia.

Também desde a versão 4.6, o Max permite a programação em Java e Javascript dentro dos próprios *patches*, o que veio alargar enormemente as possibilidades, uma vez que há procedimentos – principalmente os relativos a estruturas de repetição – que são mais fáceis de implementar em programação por código, e tem ainda embutida uma liguagem simples de *scripting* para criar, mover e ligar objectos, enviar mensagens remotas, etc.

2.2.1. Extensões

Tal como em todas as linguagens de programação, as funcionalidades de base do ambiente de programação Max podem ser expandidas através da adição de novas bibliotecas ou objectos isolados, que podem vir aperfeiçoar funções já existentes, ou podem adicionar funcionalidades completamente novas.

2.2.2. Extensões MSP

A extensão MSP, é um conjunto de objectos dedicados à análise, síntese e processamento de sinal áudio. Tendo o Max surgido num ambiente dedicado à performance e composição musical, a sua finalidade inicial foi a de permitir programar sistemas para a performance ao vivo e controlar dispositivos musicais externos, por comunicação MIDI. A evolução da capacidade de processamento dos computadores pessoais tornou possível a criação de ferramentas para o processamento de sinal áudio, bastante mais exigente computacionalmente que o processamento de mensagens MIDI.

As funções dos objectos MSP vão desde operações básicas como a leitura e gravação de ficheiros áudio, o processamento de efeitos como *delay*, *reverb*, filtros, modelos de síntese de som até operações mais complexas como síntese granular, FFT em tempo real, síntese por modelos físicos, entre outros.

O funcionamento MSP implica toda uma arquitectura de processamento de dados à velocidade áudio, que, ao contrário da comunicação de mensagens e eventos Max – que só são processados quando chamados – funciona continuamente, desde que é ligado o sistema áudio do Max, que faz a comunicação com a placa de áudio do computador.

Os objectos MSP distinguem-se graficamente dos objectos normais do Max por terem sempre um ~ no final do nome das funções (exemplo: *gate~*), e os *patchcords* que interligam objectos MSP são tracejados.

2.2.3. Extensões Jitter

Jitter é uma colecção de objectos para o processamento de dados em formato de matrizes, altamente optimizado para processamento de vídeo, gráficos e gráficos 3D.
Todos os nomes dos objectos Jitter começam por *jit.* (exemplo: *jit.xfade*). Os patchcords são também distintos dos do Max e MSP.

2.2.4. Extensões criadas pelo próprio utilizador

Para além das bibliotecas de objectos mencionadas, o Max permite também que facilmente o utilizador crie as suas próprias colecções de objectos, disponibilizando para isso os seguintes recursos:

- *patch* - programa gráfico construído na interface do Max, que requer o software para ser executado.
- *subpatch* – patch utilizado dentro de um objecto *p* ou *patcher*, que permite que um fragmento de um patch seja "encapsulado" para uma melhor arrumação, repetição e organização das várias secções de um programa.
- *abstraction* – patch programado em Max mas utilizado por outro patch, na forma de um objecto de função. Uma vez que é também ele um patch programado em Max, permite que a qualquer momento o utilizador o abra e faça alterações. Estas alterações reflectir-se-ão em todas as alegorias a esta abstracção
- *external* – função externa programada na linguagem *C*. Tal como as abstracções, são utilizadas em *patches* como objectos de função externos. No entanto, por terem sido programados fora do Max e compilados, estes não podem ser abertos e editados em Max.

São ainda disponibilizados ao utilizador dois métodos para a criação das suas próprias aplicações:

- *Application* – procedimento para a criação de aplicações, que transforma um patch Max numa aplicação fechada e independente. Uma aplicação criada neste formato já inclui (internamente) o leitor *Max Runtime*, necessário para correr o programa.
- *Collective* – solução híbrida entre o patch e a aplicação. É um programa fechado, não editável, mas que necessita do *Max Runtime* (ver abaixo) para ser executado. Um programa exportado neste formato tem a vantagem de ser muito mais pequeno, comparativamente à opção application, uma vez que o leitor necessário para a execução do programa não fica incluído no ficheiro.

2.2.5. Outras extensões existentes

Além das extensões MSP e Jitter, criadas e comercializadas pela própria Cycling '74, existem várias outras que abordam diversas funcionalidades, criadas por vários programadores por todo o mundo, e são na maioria gratuitas.

Abaixo serão ilustradas algumas das mais pertinentes para a presente dissertação.

cv.jit
(http://www.iamas.ac.jp/~jovan02/cv/)

A colecção *cv.jit*, de *Jean-Marc Pelletier*, é uma implementação em Max da biblioteca OpenCV, uma biblioteca de visão computacional open source escrita em C e C++ para vários sistemas operativos (http://opencv.willowgarage.com/wiki/, acedido em 28 de Setembro de 2009). É uma extensão da colecção Jitter com cerca de 60 objectos, especificamente criados para a implementação de algoritmos de análise de imagem.

Nota: Alguns dos objectos b.blocks utilizam objectos cv.jit, pelo que, para serem utilizados, esta biblioteca tem de estar instalada

Soft VNS
(http://homepage.mac.com/davidrokeby/softVNS.html)

Criado pelo artista David Rokeby, o *Soft VNS* é uma biblioteca altamente especializada em análise e processamento de vídeo, inicialmente desenvolvida para a sua instalação *Very Nervous System*, de 1986.

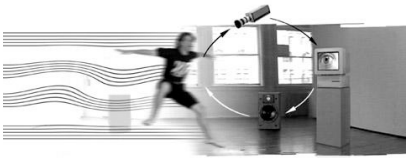


Fig. 2-6 *David Rokeby*: *Very Nervous System*

Jamoma
(http://jamoma.org/)

Jamoma é uma plataforma estruturada de desenvolvimento e controlo de módulos de alto nível no ambiente Max (Place T. , Lossius, Jensenius, & Peters, 2008) (Place & Lossius, 2006).

Desenvolvido inicialmente por *Trond Lossius* (www.trondlossius.no) e *Timothy Place* (http://community.electrotap.com/tim/), foi apresentada em 2006, com o objectivo de criar um standard para a programação e partilha de *patches*. Desde então evoluiu para uma plataforma para a pesquisa e performance em arte interactiva, que consiste em quatro ferramentas:

- Jamoma Modular – a biblioteca original de módulos

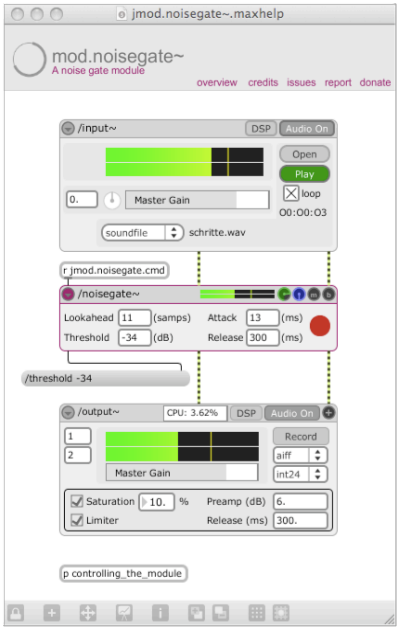


Fig. 2-7 Módulos Jamoma

para Max;

- Jamoma DSP – um interface para a programação de aplicações em C++, orientada para processamento de áudio em tempo real;
- Jamoma Multicore – uma extensão para o Jamoma DSP para a criação de representações topográficas de áudio;
- Jamoma Tools – uma colecção de utilitários para automatizar o processo de desenvolvimento e distribuição de projectos em Max, entre outros;

Este projecto distingue-se dos anteriores exemplos por ser (entre outras coisas) uma biblioteca de patches com interface gráfica, e por isso ser provavelmente o melhor exemplo para ilustrar a necessidade por parte dos utilizadores Max, de desenvolver ferramentas de mais alto nível em ambiente Max.

Integra
(www.integralive.org)

Integra é um projecto em desenvolvimento no Conservatório de Birmingham (http://www.conservatoire.bcu.ac.uk/) com o objectivo de criar um novo software para a composição e a performance musical com electrónica em tempo real.

Não se trata assim de uma biblioteca ou objecto para Max, mas de uma aplicação, programada em Max/MSP.

É uma interface gráfica que permite, de uma forma muito intuitiva, criar graficamente um patch, adicionando componentes e fazendo ligações entre eles. Cada elemento tem um painel de controlo que surge sempre na mesma posição à esquerda, quando se selecciona o módulo correspondente na área principal.

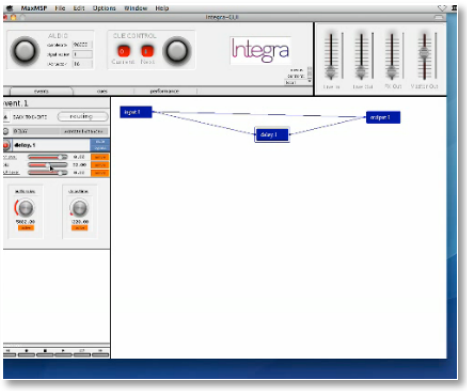


Fig. 2-8 A janela de edição de *Integra*

Sybil
(http://www.hud.ac.uk/mh/music/sybil/index.htm)

Sybil é a sigla para *Synthesis By Interactive Learning*. Trata-se de uma ferramenta interactiva para assistir

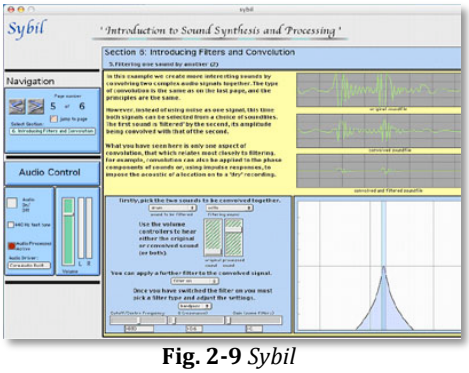


Fig. 2-9 *Sybil*

ao o ensino da síntese e processamento de som (Clarke, Watkins, Adkins, & Bokowiec, 2004).

Sybil é composto por módulos desenvolvidos em Max/MSP e disponibilizados como pequenas aplicações independentes, cada uma dedicada a um tema específico na área de síntese de som ou composição electroacústica.

2.3.Outros ambientes gráficos de programação

Apesar de ser o mais desenvolvido, o Max não é o único ambiente gráfico de programação para desenvolvimento de aplicações multimédia interactivas no panorama actual. Os programas abaixo ilustrados inserem-se no mesmo campo de acção do Max e têm abordagens muito semelhantes.

Pd (Pure Data)
(http://puredata.info/)

O Pd (*Pure Data*) é um ambiente gráfico de programação desenvolvido originalmente por Miller Puckette – o criador original do Max. É em muitos aspectos semelhante ao Max, e compartilha inclusivamente alguns objectos e *patches* comuns. Ao contrário do Max, no entanto, o Pd é de utilização livre e gratuita, e é desenvolvido por uma comunidade de programadores, seguindo os princípios do software livre. O código de base é ainda mantido por Puckette.

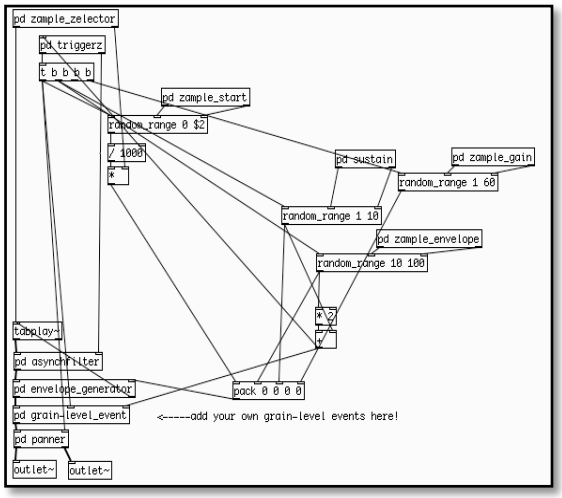


Fig. 2-10 Pure Data

Tal como o Max, foi Inicialmente concebido para a área da música por computador, e mais recentemente expandido para conteúdos multimédia.

O facto de ser de distribuição gratuita torna-o particularmente atractivo para principiantes ou para a utilizações de curta duração, e o facto de ser de código aberto é um factor especialmente interessante para uma comunidade de programadores que sejam capazes de tirar partido, expandir e melhorar a distribuição original.

No entanto, ser gratuito e aberto tem contrapartidas importantes, nomeadamente no que diz respeito à documentação e usabilidade, o que, comparativamente ao Max, o torna um ambiente menos amigável e acessível.

Nota: Para efeitos desta dissertação, foi ponderada a ideia do protótipo ser desenvolvido em Pd, pelo facto de não ser software comercial. Contudo, pesando os prós e contras entre o

software livre por um lado e o software comercial, muito melhor documentado e com uma interface bastante mais fácil de usar, apesar das aparentes semelhanças, pareceu-me mais apropriado optar pelo Max.

Mais ainda, numa perspectiva didáctica, bastante pertinente para este projecto, a minha experiência lectiva com alunos que não são de áreas de engenharia informática, e que vêm maioritariamente de áreas artísticas, nomeadamente da música por computador, artes digitais, produção audiovisual e multimédia - que são aliás bastante representativos do tipo de utilizadores que constituem a comunidade Max - tem revelado que a facilidade de utilização do Max, na perspectiva da ergonomia da interface (especialmente desde o lançamento da versão 5), bem como a excelente documentação e sistema de ajuda , são elementos de grande importância.

Isadora

(http://troikatronix.com/isadora.html)

O ambiente gráfico Isadora, da empresa *Troikatronix* (troikatronix.com), desenvolvido por *Mark Coniglio*, foi inicialmente criado para o trabalho desenvolvido com a companhia de dança *Troika Ranch* (www.troikaranch.org), da coreógrafa *Dawn Stoppiello*.

Tendo partido de um intenso trabalho voltado para a performance artística com novos media, o ambiente Isadora está extremamente bem pensado para esse fim, e especialmente optimizado para a manipulação de vídeo digital em tempo real. A forma como o ambiente está organizado permite que muito facilmente se possa fazer mudanças de *patches* entre, por exemplo, cenas diferentes numa performance. Esta é curiosamente uma situação que nos outros ambientes pode ser por vezes bastante problemática.

Disponibiliza um conjunto de mais de uma centena de módulos (*actors*), que podem ser combinados em patches na janela de edição principal (*stage*). Estes módulos têm uma representação visual dos seus parâmetros e permitem a alteração dos mesmos directamente no módulo, pelo que

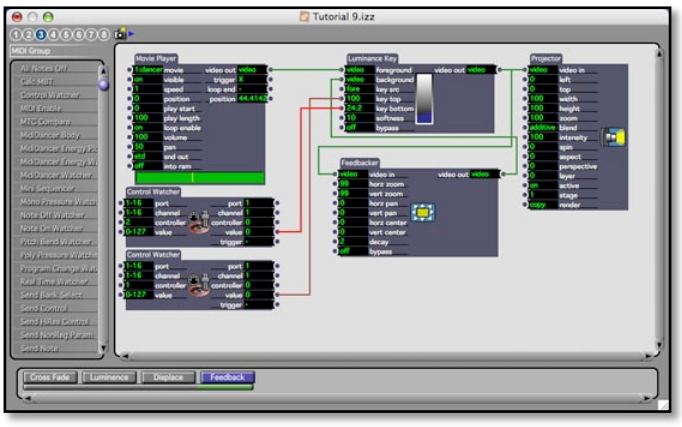


Fig. 2-11 Janela de edição do programa *Isadora*

a sua utilização é bastante fácil.

Ao contrário do Max e do Pd, o Isadora não disponibiliza muitos recursos de baixo nível, não sendo, por isso, tão flexível. Por utilizar blocos de funções pré-definidas com interface gráfica, o Isadora foi uma das referências principais para a concepção da biblioteca *b.blocks*.

EyesWeb

(http://www.infomus.org/EyesWeb/EywPlatform.html)

A plataforma *EyesWeb* foi desenvolvida no âmbito dos projectos de investigação em sistemas multimodais interactivos no *InfoMus Lab* – Laboratório de informática musical, da Universidade de Génova, em Itália. É utilizado em muitas aplicações e projectos de carácter científico, bem como para performance e instalações interactivas, e implementa sofisticados algoritmos de análise de imagem.

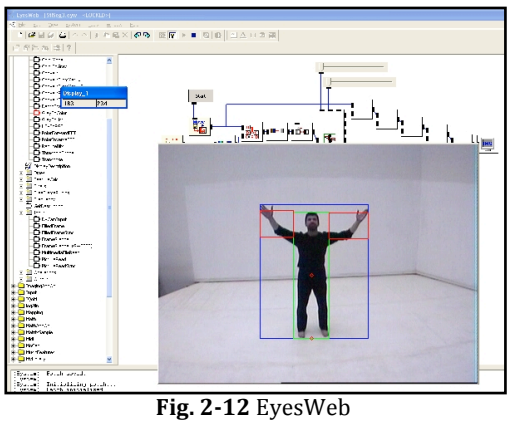


Fig. 2-12 EyesWeb

3. IMPLEMENTAÇÃO: a biblioteca *Max Building Blocks*

3.1. Introdução

A colecção *Max Building Blocks* (abreviada para *b.blocks*) consiste numa biblioteca de abstracções para o ambiente de programação Max. Algumas abstracções apresentam soluções para determinados procedimentos e funcionalidades complexas, enquanto outras são basicamente interfaces gráficas para objectos já existentes.

A biblioteca está dividida por áreas de utilização, orientadas para algumas das funções frequentemente utilizadas para contextos multimédia interactivos, sendo: análise e processamento de imagem; análise e processamento de som; comunicação; funções globais.

Ao contrário de uma solução terminada e hermética, pretende-se que esta biblioteca seja dinâmica, expansível e abrangente para além do que é apresentado para a finalidade desta dissertação. Contudo, o presente estado deve constituir já uma boa base de trabalho, suficientemente robusta e flexível para ser utilizada a curto prazo, e lançar as bases para futuros aperfeiçoamentos e expansões.

3.2. Convenções gerais

Tendo como objectivo uma utilização e integração fácil da biblioteca em *patches* com outros objectos, quer por utilizadores mais experientes como por utilizadores menos experientes, os blocos são construídos usando, sempre que possível, convenções nativas do ambiente Max. Assim, e também por uma questão de coerência e abertura, todos os termos, funções, parâmetros, descrições e documentação, estão em inglês.

Todas as abstracções *b.blocks* são nomeados de acordo com a função que desempenham, e sempre precedidos por "b.", assim como, por exemplo, os objectos da biblioteca Jitter são sempre precedidos por "jit.".

Para efeitos desta dissertação e protótipo, a designação módulo refere-se a uma abstracção com interface gráfico. Ou seja, um *patch* programado em Max para ser instanciado dentro

de outros *patches*, tal como uma *abstaction* (ver 2.2.5 - Outras extensões existentes), mas em *bpatchers*.

3.3. Arquitectura

Cada abstracção consiste num patch Max pré-programado que, tal como uma abstracção, está preparado para ser usado dentro de outros patches. No entanto, tal como foi explicado acima (ver 3.2: Convenções gerais), ao contrário das abstrações e objectos externos, utilizados em caixas de objecto normais, as abstracções *b.blocks* são programados para serem utilizados dentro de objectos *bpatcher*, que permitem aceder e visualizar o interior do patch associado.

Este é um aspecto essencial da biblioteca *b.blocks*, uma vez que, para além de disponibilizar as funções, tem como objectivo disponibilizar uma interface gráfica para as mesmas, de forma a que o utilizador possa utilizar as funções de forma rápida e intuitiva.

Um módulo *b.blocks* surge num patch Max como um pequeno objecto rectangular de cor cinzenta. Cada um tem uma aparência própria, uma vez que o tipo de controlos e mostradores utilizados dependem dos parâmetros associados à sua função específica.

Fig. 3-1 Módulo *b.blocks*

3.3.1. Constituição de um módulo

Cada módulo é constituído por duas secções principais: o cabeçalho e a área dos controlos para os parâmetros. Tal como é convenção em Max, em cima situam-se as entradas (*inlets*) e em baixo as saídas (*outlets*).

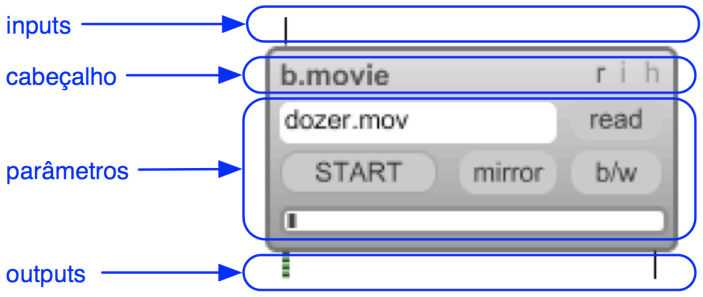


Fig. 3-2 Secções de um módulo

Cabeçalho

O cabeçalho contém duas áreas, sendo a da esquerda destinada ao nome do bloco, e a da direita reservada três pequenos botões, comuns a todos os módulos, definidos pela letra inicial da função a que se destinam (em inglês).

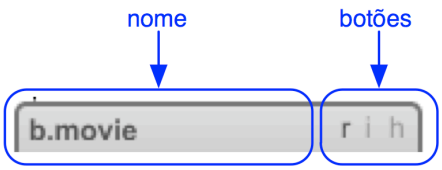


Fig. 3-3 Cabeçalho

Assim, a letra **r** (*remote*), activada por defeito, diz respeito ao controlo dos parâmetros através de outros módulos ou mensagens de controlo enviadas remotamente. Se este botão estiver desactivado, o módulo não recebe mensagens remotas de controlo. O botão **i** (*info*), permite visualizar uma lista com os inputs, outputs e mensagens que o módulo reconhece. O botão **h** (*help*), dá acesso ao patch de ajuda do módulo.

Corpo

A área do corpo do módulo contém todos os controlos e indicadores relativos aos parâmetros específicos do módulo.

Os módulos estão concebidos de forma a que o utilizador possa aceder directamente aos parâmetros necessários para executar a função a que cada módulo se destina. No entanto, além dos controlos visíveis, alguns módulos permitem aceder a outras funções, através de mensagens enviadas pelos *inlets*.

3.3.2. Patch interno

Uma vez que a colecção foi inteiramente programada em Max, cada módulo é um patch Max que, apesar de programado para ser usado dentro de um *bpatcher*, pode também ser

instanciado dentro de um objecto **p** (*patcher*), ou aberto directamente no ficheiro correspondente.

Os módulos foram programados tendo em conta que num mesmo patch podem ser instanciados mais de uma vez. Para tal, não foram utilizados objectos *send* e *receive*, para envio de mensagens internas, uma vez que isso faria com que um valor ou mensagem fosse recebida por todos os módulos semelhantes. Como resultado, os patches podem por vezes ter vários *patchcords* que não puderam ser evitados. Apesar disto, foi sempre dada especial atenção a que os *patches* se mantivessem o mais claro e lógicos quanto possível. Para isso, e apesar das diferenças entre os vários *patches*, a estrutura geral é semelhante e podem ser reconhecidos facilmente alguns pontos comuns.

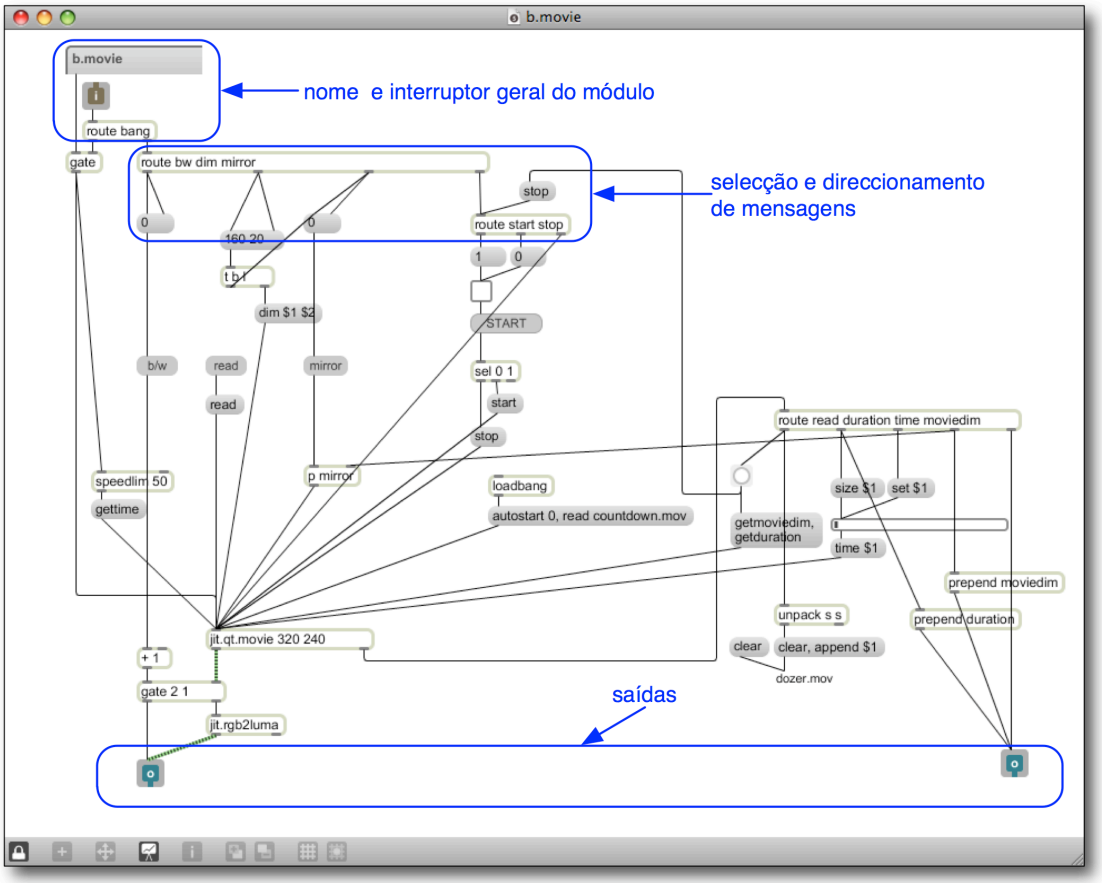


Fig. 3-4 Interior de um patch b.blocks

Na parte superior do patch situam-se as entradas, o nome do módulo e um interruptor geral que o desactiva. O objecto com o nome é ele próprio uma abstracção chamada **b.header**, instanciada dentro de um *bpatcher*. O b.header surge em todos os patches *b.blocks*, e define o nome e tamanho de cada um. Além disso, inclui também os botões que se encontram do lado direito do cabeçalho. Ao serem instanciados, são utilizados três argumentos nas

propriedades do bpatcher. O primeiro argumento define o nome do patch a instanciar, enquanto o segundo e terceiro argumentos definem, respectivamente, o seu tamanho horizontal e vertical.

Desta forma a aparência é mantida coerente em toda a biblioteca, e facilita o desenvolvimento e actualizações uma vez que qualquer alteração feita no ficheiro global b.header se vai reflectir em todos os módulos.

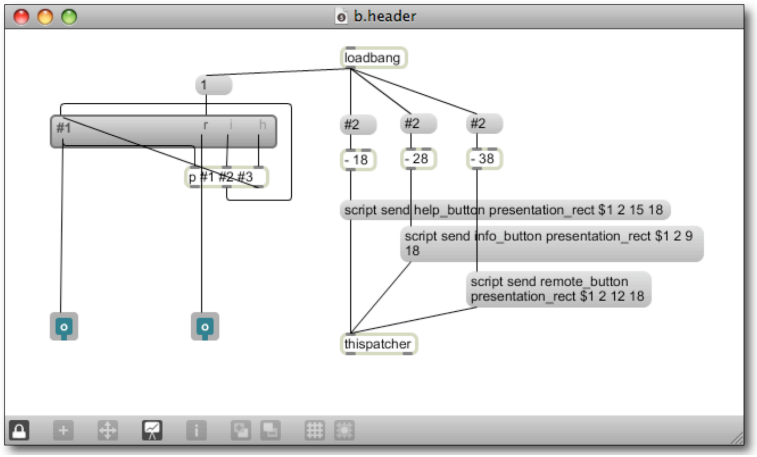


Fig. 3-5 O módulo *b.header*

O painel cinza à esquerda serve de *background* ao módulo e o seu tamanho é ajustados automaticamente para cada um. Sobrepostos estão quatro botões. À esquerda está o #1, que no momento em que o módulo é criado, assume o nome do patch instanciado. Grande parte dos módulos podem ser activados ou desactivados com um clique no seu nome. À direita encontram-se os três botões explicados atrás.

O *subpatch* "p #1 #2 #3" que se vê por baixo dos três botões (r-i-h) executa as acções controladas pelos botões i e h, dando acesso, respectivamente, à janela flutuante de informações sobre os inputs, outputs e mensagens reconhecidas pelo módulo, e ao patch de ajuda (*help*).

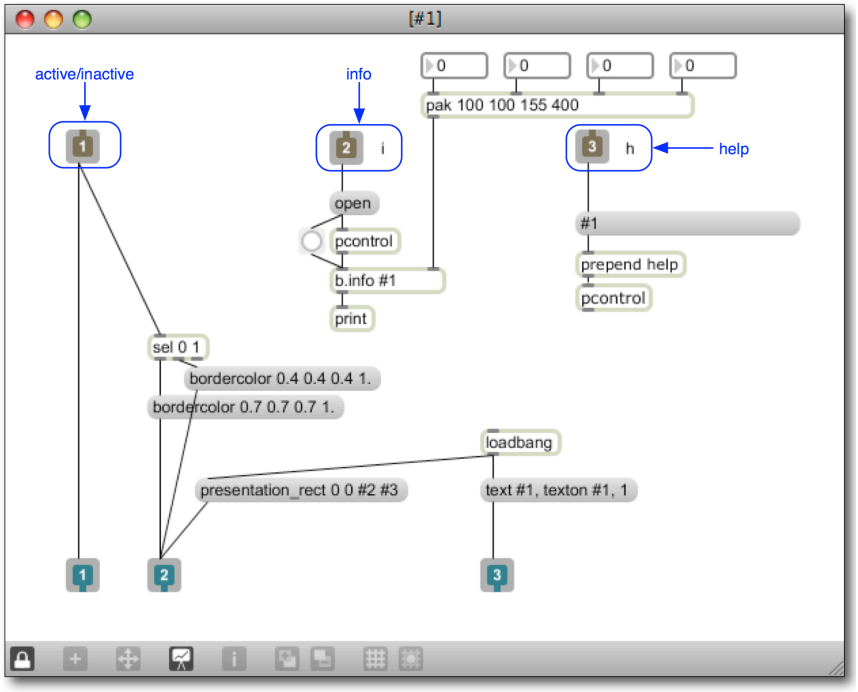


Fig. 3-6 *Subpatch*

Os três *inlets*, aqui marcados a azul, recebem as ordens enviadas pelo botão do nome do módulo e pelos botões **i** - **h**.

Desde o lançamento, em Maio de 2008, da versão 5 do software, o Max tem dois modos de visualização completamente independentes: o **modo de edição**, já existente nas versões anteriores, e o **modo de apresentação**, completamente novo. O modo de edição é o modo utilizado para a programação normal do patch, onde estão as ligações (*patchcords*) e todos os objectos que constituem o patch. O modo de apresentação permite ver apenas os objectos seleccionados para esta visualização, sem qualquer tipo de ligação entre eles.

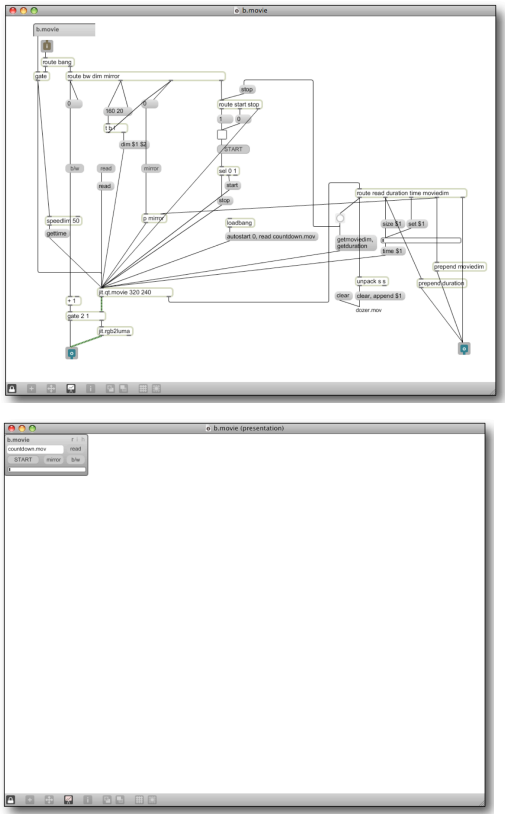


Fig. 3-7 Modo de edição (em cima) e modo de apresentação (em baixo).

Isto permite que seja possível construir o *patch* sem preocupações relativamente ao *layout* final, uma vez que, uma vez seleccionados para aparecerem no modo de apresentação, os objectos podem ser reposicionados e escalados

à vontade, sem qualquer interferência no funcionamento do *patch*.

A biblioteca *b.blocks* utiliza amplamente esta funcionalidade. Cada módulo tem os seus objectos de interface visíveis no modo de apresentação e o *patch* organizado de forma lógica, de acordo com os objectos presentes e respeitando, sempre que possível, a lógica do fluxo de dados.

3.4. Documentação

3.4.1. Sistema de ajuda

Tal como todos os objectos da distribuição Max, todos os *patches b.blocks* têm um patch correspondente de ajuda. Os *patches* de ajuda são programas funcionais, que demonstram todas ou as principais características e funções de cada módulo, e que, tal como na ajuda dos objectos Max, podem ser utilizados, alterados e copiados.

Os *patches* de ajuda são acedidos usando o pequeno botão **h** (*help*), que se encontra no canto superior direito de todos os módulos.

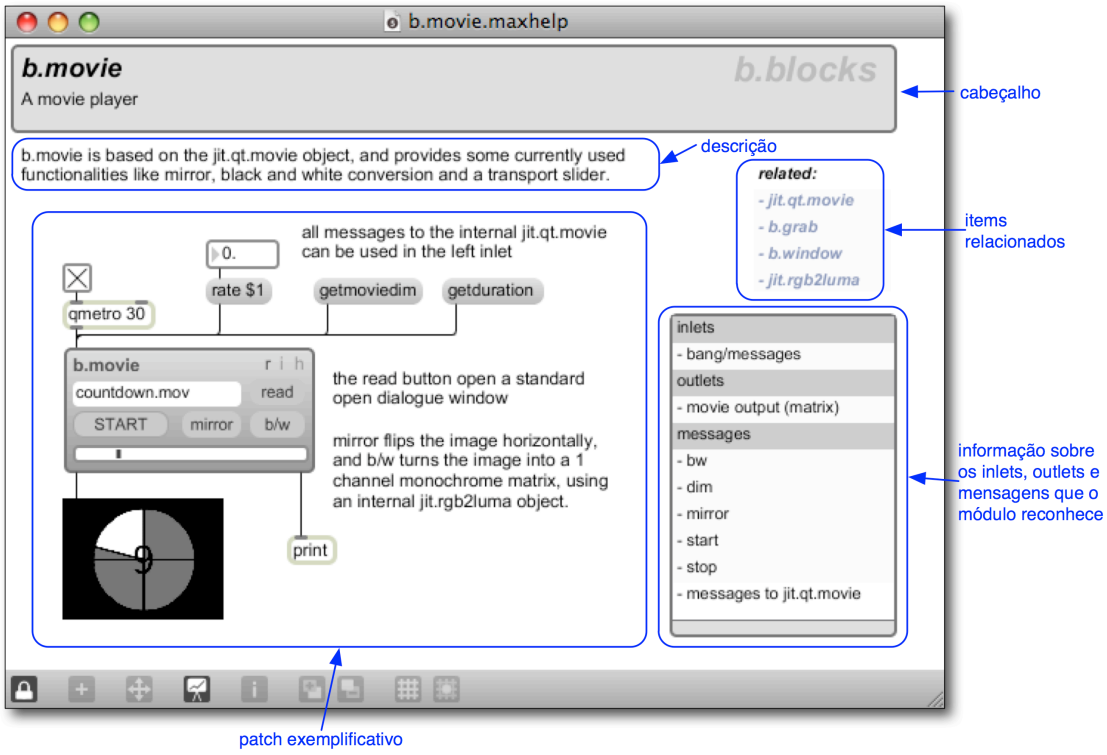


Fig. 3-8 Organização de um patch de ajuda

Na parte superior encontra-se o cabeçalho, com o nome do módulo e uma descrição muito sucinta da sua função. Logo abaixo surge uma descrição mais detalhada. A área maior abaixo da descrição é onde está o patch exemplificativo das funções principais. No lado direito encontram-se uma pequena lista com itens relacionados, que podem ser objectos, outros módulos ou *patches* relevantes. Os itens podem ser usados directamente como links

de acesso directo ao ficheiro de ajuda do item correspondente. Abaixo deste encontra-se uma lista com informação dos *inlets*, *outlets* e mensagens que o módulo reconhece.

3.4.2. Exemplos

De modo a facilitar a aprendizagem da colecção, foram criados *patches* exemplificativos, que demonstram o funcionamento básico de alguns dos módulos e tentam, simultaneamente, ilustrar algumas das áreas abrangidas pela colecção e servir como tutoriais para as primeiras utilizações da biblioteca.

3.5. Módulos

Os módulos b.blocks estão divididos em categorias, seleccionadas por abrangerem um vasto leque de possibilidades, geralmente associadas aos sistemas digitais multimédia interactivos.

As categorias são:

- Imagem (processamento/análise);
- áudio (processamento/análise);
- comunicação (inclui controladores externos);
- funções globais;

Além dos objectos nativos do Max, MSP e Jitter, foram utilizados alguns objectos de bibliotecas externas, nomeadamente:

- biblioteca *cv.jit* (*Jean-Marc Pelletier*, http://www.iamas.ac.jp/~jovan02/cv/)
- CNMAT (http://cnmat.berkeley.edu/downloads)
- Objecto *aka.wiiremote* (*Masayuki Akamatsu*, http://www.iamas.ac.jp/~aka/max/)

3.5.1. Processamento de Imagem

---

b.brcosa Interface para o objecto *jit.brcosa*. Permite ajustar o brilho (*brightness*), contraste (*contrast*) e saturação (*saturation*) da imagem.

---

b.colorfilter Deixa passar apenas a cor seleccionada ou todas as cores menos a seleccionada.

---

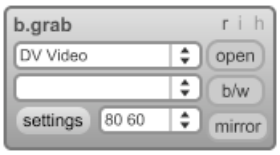
b.crop Corta tudo o que está fora da região seleccionada.

---

b.framediff Subtracção de cada frame pela frame anterior.

---

b.grab Interface para o objecto *jit.qt.grab*.
Capta a imagem de uma câmara de vídeo que esteja ligada ao computador.
Permite definir a resolução



---

b.movie Interface para o objecto *jit.qt.movie*.
Reprodução de qualquer ficheiro de vídeo suportado pelo *Quicktime*.

---

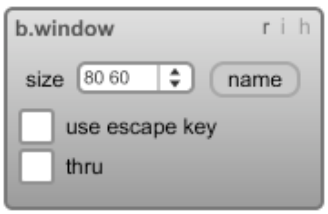
b.negative Inverte as cores da imagem de entrada para o seu negativo.

---

b.videomixer Mistura de duas entradas de vídeo A e B.
Permite definir o operador.

---

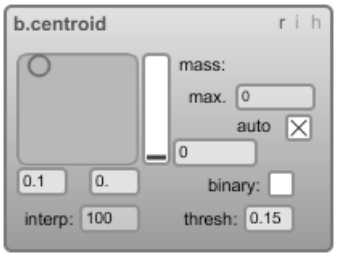
b.window Interface para o objecto *jit.window*.
Cria uma janela e envia para lá o vídeo de entrada. Permite definir o tamanho e o nome da janela, e mostrar o vídeo em ecrã inteiro (*fullscreen*).



3.5.2. Análise de Imagem

---

b.centroid Interface para o objecto *cv.jit.centroids*, da biblioteca ***cv.jit***.
Mede o brilho em todos os pixéis de cada frame para determinar as coordenadas de um ponto, que corresponde à média desses valores. Calcula também a soma do brilho de todos os pixéis em cada frame (*mass*).



---

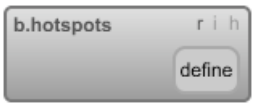
b.colortrack Detecta o ponto médio de uma determinada cor em cada frame.
Permite definir uma tolerância e fazer interpolação.

---

b.features Interface para o objecto *cv.jit.features*, da biblioteca ***cv.jit***.
Determina pontos de alto contraste na imagem.

---

b.hotspots Permite definir até 8 áreas de tamanhos diferentes num plano bidimensional.



---

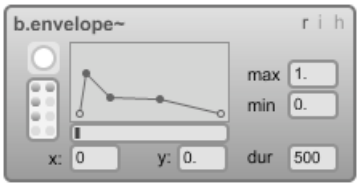
b.presence Permite capturar uma frame para fazer subtracção de *background* e detecção de presença.
Pode adaptar-se dinamicamente.

3.5.3. Processamento de som

Os objectos de processamento de som são construídos com a biblioteca MSP, e tal como os objectos MSP, os módulos b.blocks de áudio têm um ~ no fim do nome.

b.brickwall~ Filtro passa-banda FFT de corte abrupto nos pontos definidos. Os valores são definidos em bandas de frequência (*bins*).

b.envelope~ Gerador de envolvente. Pode ser usado em áudio e/ou em dados.

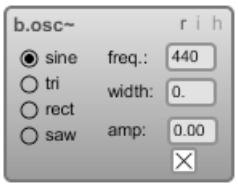


b.freqsplit~ Cadeia de filtros FFT para separar um sinal áudio em 2, 3 ou 4 regiões de frequências diferentes, definidas pelos *splitpooints*. Os valores são definidos em bandas de frequência (*bins*).

b.gain~ Controlo de ganho com dois canais emparelhados ou independentes.

b.osc~ Gerador de sinal com quatro tipos de onda diferentes (sinusoidal, triangular, quadrada e dentes de serra).

A amplitude pode ser controlada interna ou externamente.



---

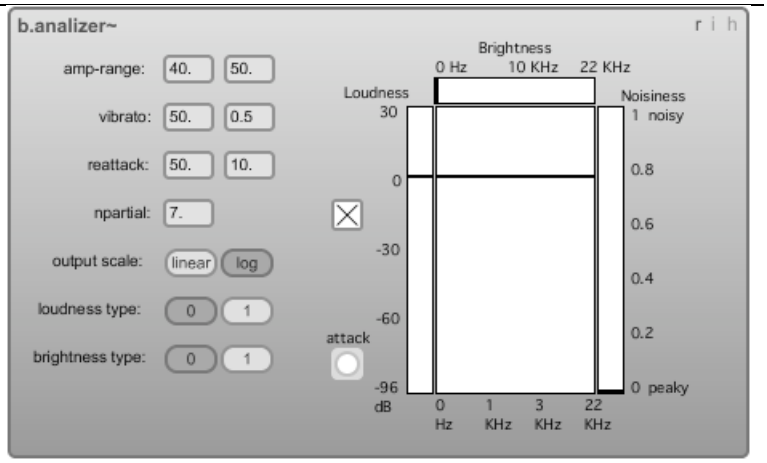
b.pan~ Distribuidor de dois canais para duas saídas (panorâmica).

3.5.4. Análise de Som

---

b.analizer~

Interface para o objecto analizer~ de Tristan Jehan, integrado na biblioteca desenvolvida na CNMAT, em Berkeley (EUA).

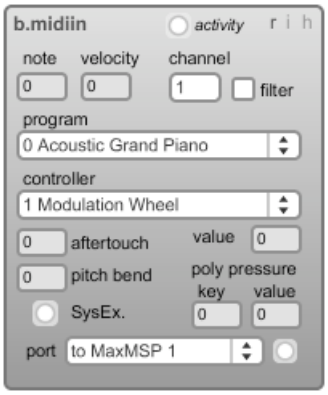


3.5.5. Comunicação

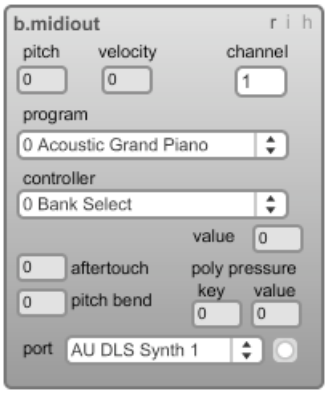
Este grupo de módulos contempla comunicação MIDI (Musical Instrument Digital Interface), e comunicação com controladores externos.

b.hi Interface para o objecto *hi* (*human interface*), para dispositivos de interface externos.

b.midiin Visualizador de entrada de todo o tipo de mensagens MIDI (Musical Instrument Digital Interface).
Permite filtrar a entrada por porta e por canal.



b.midiout Permite o envio de vários tipos de mensagens MIDI.



b.wacom Interface para o objecto *wacom*, de *Jean-Michel Couturier*, incluído na biblioteca CNMAT, para receber dados de uma mesa digitalizadora Wacom.

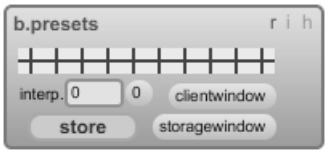
b.wiiremote Interface para o objecto *aka.wiiremote*, () para comunicação com o controlador *bluetooth Wii Remote*, da Nintendo.

3.5.6. Funções globais

Estas funções não estão associadas a nenhum contexto específico.

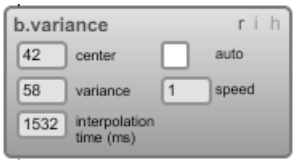
---

b.presets Gestão e armazenamento global de memórias do estado de todos os objectos *b.blocks* presentes no patch.



---

b.variance Gera números aleatórios à volta de um valor central, com interpolação. Os números podem ser gerados de forma manual ou automática.



---

4. TESTES

Durante o período em que decorreu a construção do protótipo b.blocks para esta dissertação, foram disponibilizados alguns dos módulos, em situações distintas.

ESAD Matosinhos

A primeira foi na Escola de Artes e Design de Matosinhos, onde leccionei a disciplina Atelier, aos alunos do 4º ano do curso de Artes Digitais e Multimédia. Foi apresentada em Março de 2009 uma instalação interactiva pelos alunos Guilherme Gomes, Leonardo Guedes, Manuel Fardilha e Ricardo Gomes, programada em Max/MSP/Jitter, onde foram usados alguns dos objectos b.blocks, para análise e processamento de vídeo.

Fig. 4-1 *Polligital*. Ficha técnica e imagens

ACERT - Tondela

Em Maio 2009 dei um pequeno workshop de apenas um dia e meio, na associação ACERT, em Tondela, onde três participantes, José Tavares, José Marques e Cláudio Alves, profissionais da área do espectáculo e dos audiovisuais, puderam experimentar alguns dos módulos. Depois de terem uma pequena introdução geral ao Max, disponibilizei os módulos, sem dar praticamente nenhuma explicação do seu funcionamento. Até ao fim do workshop os participantes fizeram os protótipos para duas instalações interactivas, sendo uma com desenho através de detecção de cor, e uma com captação de vídeo directo, processado em tempo real.

Fig. 4-2 ACERT - protótipo com processamento de vídeo

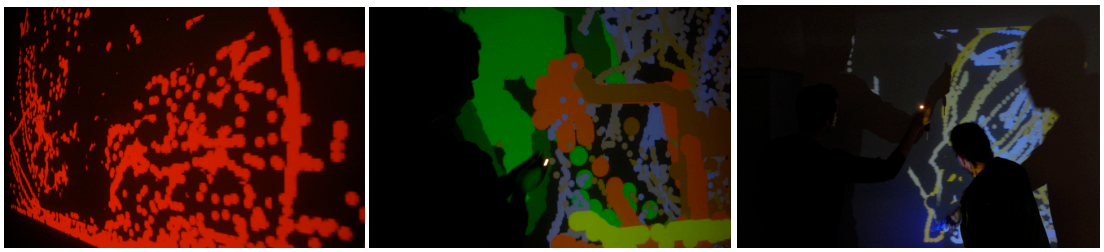


Fig. 4-3 ACERT - Protótipo com gráficos em tempo real

Como exemplo ilustrativo da utilização dos objectos b.blocks, na Fig. está um dos *patches* criados pelos participantes neste workshop, seguido de um *patch* – programado por mim para efeitos desta dissertação – que executa exactamente as mesmas funções, sem a utilização dos objectos b.blocks.

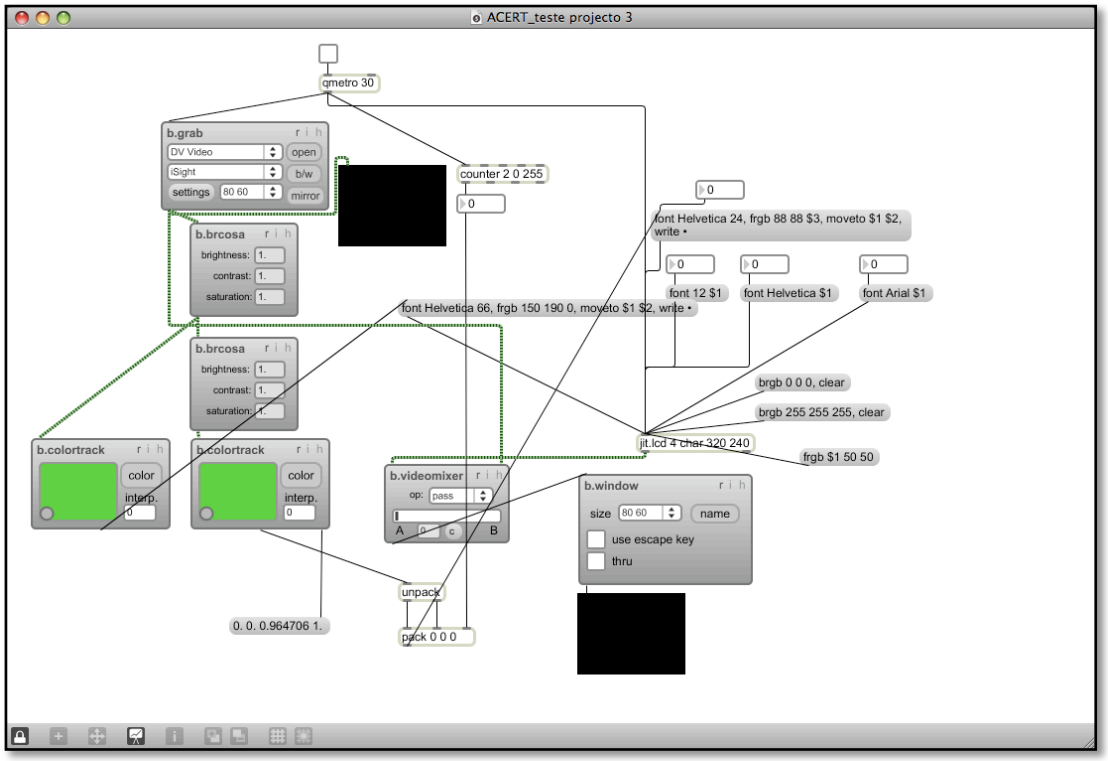


Fig. 4-4 ACERT - *patch* de geração interactiva de gráficos.

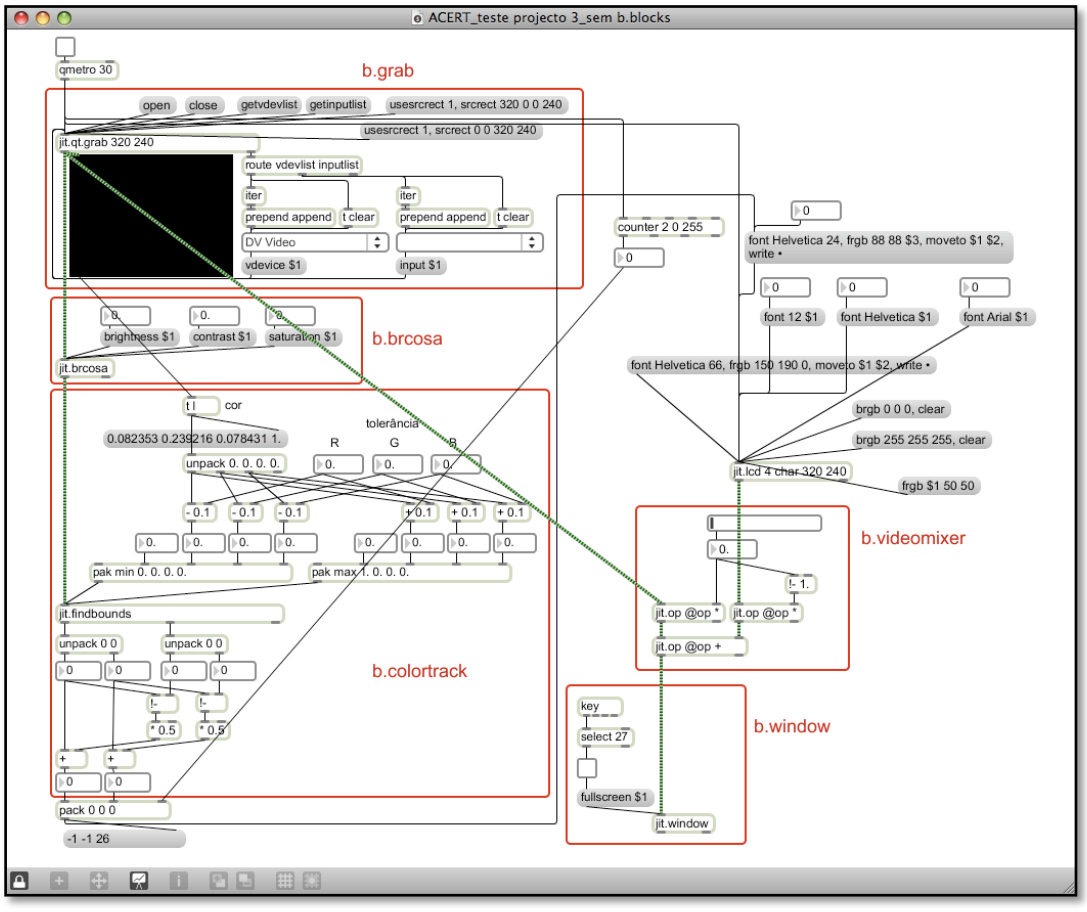


Fig. 4-5 ACERT- reprodução do *patch* da Fig. 4-4 sem utilização dos b.blocks.

Os rectângulos vermelhos mostram as secções do *patch* que correspondem aos módulos b.blocks utilizados.

Com a utilização dos b.blocks, a leitura do patch torna-se consideravelmente mais simples, nomeadamente porque, devido às mensagens locais, que definem os parâmetros, o segundo patch fica mais confuso visualmente. Recorrendo à encapsulação (subpatches) com o objecto "*p*" (ou *patcher*), este problema facilmente poderia ser resolvido.

O aspecto mais relevante, contudo, é o facto de que várias das operações utilizadas dificilmente poderiam ter sido feitas por um utilizador com pouca experiência, e mesmo a utilização de secções encapsuladas não é tão óbvia durante o processo de desenvolvimento de um patch como é depois de concluído, pelo que também neste aspecto os b.blocks facilitam a programação, por apresentarem rotinas previamente pensadas, com os parâmetros correspondentes imediatamente acessíveis. Isto evita o tempo perdido à procura da mensagem ou formatação correcta, até para utilizadores avançados.

FEUP – Ana Leite

Alguns objectos foram ainda utilizados pela Ana Leite, minha colega no programa de Mestrado em Multimédia, para o desenvolvimento do protótipo "Pinturas Sonoras", integrante da sua dissertação.

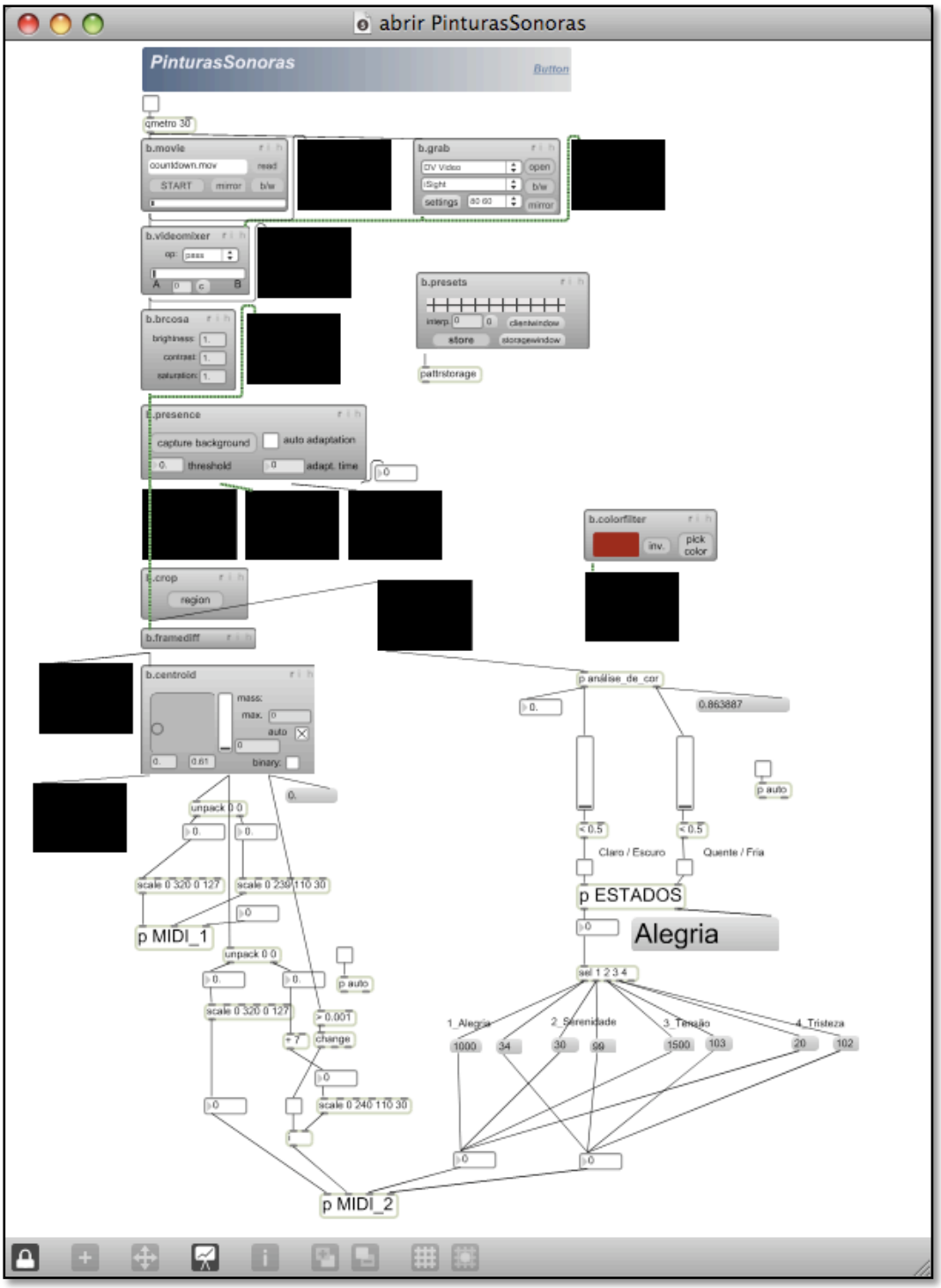


Fig. 4-6 *patch* "Pinturas Sonoras".

5. CONCLUSÕES

A afluência crescente de interesse nas novas tecnologias e novos media fazem com que o campo da multimédia e sistemas interactivos esteja em ascensão. Pela diversidade de vertentes e áreas de acção que podem ser envolvidas, tal como as inúmeras possibilidades de aplicação destes sistemas em áreas muito diversificadas, torna-se naturalmente necessário o desenvolvimento de ferramentas cada vez mais eficazes e poderosas, mas também mais versáteis e acessíveis.

A criação e desenvolvimento de ambientes gráficos, mais intuitivos e versáteis, permitiu a uma grande quantidade de utilizadores participarem activamente nesta área, e com isso trazer novas ideias, paradigmas e desafios técnicos.

Tal como o hardware, o software tem também de acompanhar as necessidades de uma comunidade de utilizadores mais vasta e frequentemente não especializada, num campo onde a complexidade dos recursos utilizados pode ser muito grande, e a atenção à facilidade de utilização, a usabilidade e ergonomia têm uma relevância cada vez maior.

Relativamente ao ambiente de programação Max, a importância que foi dada a aspectos de "cosmética" e de interface, principalmente relativa à facilidade de utilização, no desenvolvimento da versão 5 – de longe a mudança mais radical que o Max sofreu, desde a primeira versão – são reflexos das necessidades dos utilizadores relativamente a estas questões, e marca, quanto a mim, um ponto interessante de cruzamento entre a comunidade científica e a comunidade artística.

O aparecimento de recursos como o *Max Toolbox*, e o *Jamoma*, são também representativos da crescente necessidade de resolver problemas de interface e melhorar a experiência de programação em Max.

5.1. Verificações

As experimentações descritas no capítulo anterior revelaram que, mesmo em estado muito inicial, os objectos *b.blocks* cumpriram o objectivo de serem eficazes, sendo simultaneamente práticos e fáceis de utilizar e de integrar com outros objectos Max.

O *feedback* dos utilizadores deu bastantes pistas para o melhoramento dos módulos, e ideias para a construção de outros.

Verifiquei também que este é um projecto ambicioso, e, para corresponder à ideia de abrangência e facilidade de utilização, é necessário muito trabalho e muita experimentação.

5.2. Futuros desenvolvimentos

Uma biblioteca deste tipo é um trabalho contínuo de actualização e aperfeiçoamento, que deve acompanhar as evoluções do Max e o aparecimento de novas tecnologias e dispositivos que possam ser utilizados com este ambiente.

A curto prazo serão desenvolvidos mais módulos b.blocks, nomeadamente:

- Síntese de som;
- processamento de som;
- comunicação por *Open Sound Control* (http://opensoundcontrol.org/);
- comunicação com a placa *Arduíno* (www.arduino.cc);
- aperfeiçoamento do sistema de *presets*;
- aperfeiçoamento do sistema de controlo remoto por dispositivos externos;

Com o objectivo de dar seguimento de forma sólida e continuada, os seguintes aspectos serão fundamentais:

- Utilização da biblioteca por mais utilizadores, de meios e níveis de experiência diferentes. O seu *feedback* será essencial para o desenvolvimento desta biblioteca.
- Colaborações de outros programadores que possam contribuir e aperfeiçoar módulos de áreas específicas.
- Distribuição pela comunidade internacional de utilizadores Max, no fórum da Cycling'74

Coda

A interactividade está para ficar. Desde sistemas de segurança e domótica ao entretenimento e arte, estas tecnologias estão cada vez mais presentes, e o estudo da relação homem-máquina está transformar o modo como nos relacionamos com a tecnologia para uma relação mais transparente e natural.

Com a simplificação da programação para sistemas multimédia interactivos, quem sabe se veremos num futuro próximo a empresa *Apple* a lançar com o seu sistema operativo um produto neste campo. Talvez o *iInteract* ?

6. Referências

art+com. (2002). *medial stage and costume design*. Obtido em 27 de Setembro de 2009, de art+com: http://www.artcom.de/index.php?option=com_acprojects&page=6&id=29&Itemid=115&details=0&lang=en

Clarke, M., Watkins, A., Adkins, M., & Bokowiec, M. (2004). Sybil: Synthesis by Interactive Learning. *International Computer Music Conference.* Miami: ICMC.

CNMAT. (s.d.). *University of California at Berkeley CNMAT*. Obtido em 28 de Setembro de 2009, de Center for New Music and Audio Technologies (CNMAT): http://cnmat.berkeley.edu/downloads

EyesWeb Open Platform. (s.d.). Obtido em 27 de Setembro de 2009, de InfoMus Lab: http://www.infomus.org/EyesWeb/EywPlatform.html

EyesWeb Project. (s.d.). Obtido em 26 de Setembro de 2009, de InfoMus Lab: http://www.infomus.org/EywMain.html

Gomes, G. (2009). *sofisma*. Obtido em 25 de Setembro de 2009, de http://sofisma.eu/newsletter/#polligital

Max objects database. (s.d.). Obtido em 28 de Setembro de 2009, de Max objects database: http://maxobjects.com/

net, m. a. (s.d.). *media art net*. Obtido em 26 de September de 2009, de media art net: http://www.medienkunstnetz.de/mediaartnet/

Packer, R., & Jordan, K. (2002). *Multimedia - form Wagner to virtual reality.* Norton.

Place, T., & Lossius, T. (2006). Jamoma: a modular standard for structuring patches in Max. *Proceedings of the International Computer Music Conference.* New Orleans: ICMC.

Place, T., Lossius, T., Jensenius, A. R., & Peters, N. (2008). Flexible control of composite parameters in Max/MSP. *Proceedings of the International Computer Music Conference 2006.* Belfast.

Puckette, M. S. (1988). *Max reference manual* .

Pure Data Portal. (s.d.). Obtido em 26 de September de 2009, de Pure Data Portal: http://puredata.info/

Rokeby, D. (s.d.). *David Rokeby*. Obtido em 26 de Setembro de 2009, de David Rokeby: http://homepage.mac.com/davidrokeby/home.html

Troikatronix home. (s.d.). Obtido em 26 de Setembro de 2009, de Troikatronix: http://troikatronix.com/

Winkler, T. (1995). Making Motion Musical: Gesture Mapping Strategies for Interactive Computer Music. *International Computer Music Conference.*

Zicarelli, D., & Taylor, G. (2006). Max Fundamentals. Cycling'74.

Anexo

b.blocks: Manual de referência

As páginas seguintes apresentam um formato piloto para a documentação de referência detalhada de alguns dos objectos b.blocks mais desenvolvidos.

O símbolo $ é usado para representar o valor pretendido com o envio da mensagem correspondente. O valor pode ser de vários tipos:

$i -número inteiro (int);

$f – número decimal (float);

$s – texto (symbol);

Módulos de Análise e Processamento de Imagem

b.brcosa

Graphical interface for the ***jit.brcosa*** object. Adjusts the brightness, contrast and saturation of a video input.

GUI

	Name	Type	Range	Description
1	brightness	float	any	Brightness value.
2	contrast	float	any	Contrast value.
3	saturation	float	any	Saturation value.

Inlets

	Name	Type	Range	Description
1	matrix input, brightness, messages	Matrix, float, symbol	any	Input for the video matrix stream. A float will adjust the brightness value. Input for all the messages.
2	contrast	float	any	Contrast value.
3	saturation	float	any	Saturation value.

Outlets

	Name	***Type***	***Range***	***Description***
1	matrix output	Matrix		Video matrix output

Messages

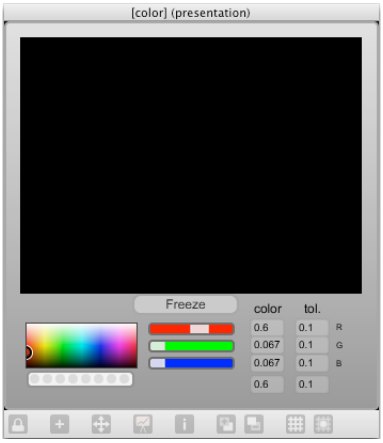
Format	Range	Description
brightness $f	any	Brightness value. Values above 1 increase brightness, while values below 1 will decrease it.
contrast $f	any	Contrast value. Values above 1 increase contrast, while values below 1 will decrease it.
Saturation $f	any	Saturation value. Values above 1 increase saturation, while values below 1 will decrease it.

b.colorfilter

b.colorfilter allows the selection of one color with a tolerance factor, and blacks out all other colors. Alternatively, the inverse button will black out only the selected color range.

GUI

	Name	Type	Range	Description
1	Inverse	toggle		
2	Pick color	toggle		Opens or closes the color picker window



	Name	Type	Range	Description
1	screen			A click on the screen will select the color in the mouse position
2	color picker			The color center value
3	presets			Color presets
4	freeze	toggle		Freezes the input image so that choosing a color is easier
5	rgb indicators			Visual indicators for the currently selected color range
6	rgb color	float	0.-1.	Color center rgb values
7	rgb tolerance	float	0.-1.	Color rgb tolerance values

Inlets

	Name	Type	Range	Description
1	Matrix, messages	Matrix		Input for the video matrix stream.

Outlets

	Name	Type	Range	Description
1	filtered output	Matrix		Video matrix output
2	selected color	list		Color values of the color selected in the colorpicker.

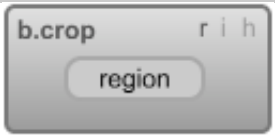
Messages

Format	***Range***	***Description***
color $f $f $f	0.-1.	Brightness value. Values above 1 increase brightness, while values below 1 will decrease it.
tol $f $f $f	0.-1.	Contrast value. Values above 1 increase contrast, while values below 1 will decrease it.

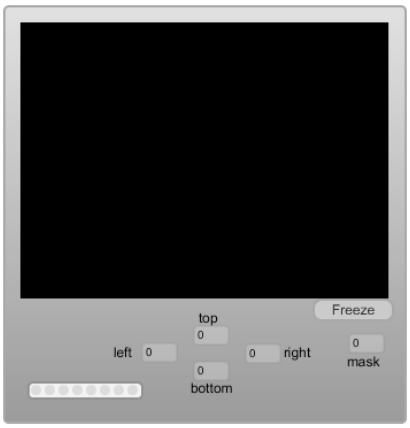
b.crop

Crops the image outside the selected area.

GUI



	Name	*Type*	*Range*	*Description*
1	region	toggle		Opens the region selection window



	Name	*Type*	*Range*	*Description*
1	screen			The input image and the selection box will be shown here
2	presets			Region presets
3	Left, top, right, bottom	int	0-?	Selection of the crop region
4	freeze	toggle		Freezes the input image
6	mask	int	0-255	Mask transparency level

Inlets

	Name	*Type*	*Range*	*Description*
1	Matrix, messages	Matrix		Input for the video matrix stream.

Outlets

	Name	*Type*	*Range*	*Description*
1	output	Matrix		Cropped output

Messages

Format	*Range*	*Description*
crop $i $i $i $i	0-?	Crop region. Four values to input the left top right and bottom coordinates.

b.framediff

Subtracts the current frame with the last frame. The output will show only the changed pixels.

GUI

Name	*Type*	*Range*	*Description*
There are no parameters. Only the name button will work.			

Inlets

	Name	*Type*	*Range*	*Description*
1	matrix input	Matrix		Input for the video matrix stream.

Outlets

	Name	*Type*	*Range*	*Description*
1	matrix output	Matrix		Video matrix output

Messages

Format	*Range*	*Description*
none		

b.grab

Graphical interface for the ***jit.qt.grab*** object. Grabs video from any quicktime compatible external digitizer.

GUI

	Name	*Type*	*Range*	*Description*
1	digitizer	Menu		
2	driver	Menu		
3	settings	Button		
4	dimensions	Menu		
5	open/close	toggle		
6	b/w	toggle		
7	mirror	toggle		

Inlets

	Name	*Type*	*Range*	*Description*
1	matrix input, messages	Matrix, symbol		Input for the video matrix stream. Input for all the messages.

Outlets

	Name	*Type*	*Range*	*Description*
1	matrix output	Matrix		Video matrix output.
2	Message output	Symbol		Messages from the internal jit.qt.grab object.

Messages

Format	*Range*	*Description*
open		Opens the selected digitizer
close		Closes the current digitizer
dim $i $i	0-?	Video matrix dimensions.
mirror $i	0/1	Flips the image horizontally
bw $i		Converts the image to black and white (luma key)
bang		Manually trigger a frame

b.grab will receive any other message to the jit.qt.grab object, so all the messages of this object can be used.

b.movie

A movie player based on the jit.qt.movie object. Reads any quicktime compatible video format.

GUI

	Name	Type	Range	Description
1	filename	label		Current file name
2	read	button		Open dialogue window
3	start/stop	toggle		Start/stop movie
4	mirror	Menu		Flip the image horizontally
5	b/w	toggle		Convert to monochrome (luminance)
6	position	slider		

Inlets

	Name	Type	Range	Description
1	bang, messages	Bang, symbol		A bang outputs one frame. Input for all the messages.

Outlets

	Name	Type	Range	Description
1	matrix output	Matrix		Video matrix output.
2	Message output	Symbol		Messages from the internal jit.qt.movie object.

Messages

Format	Range	Description
bw $i	0/1	Toggles black and white (monochrome) mode
dim $i $i	0-?	Video matrix dimensions.
mirror $i	0/1	Toggles mirror (horizontal flip) mode
start		Starts the currently loaded movie
stop		Stops the currently loaded movie

b.movie will receive any other message to the jit.qt.movie object, so all the messages of this object can be used.

b.negative

Inverts the colors to obtain the negative color image.

GUI

Name	*Type*	*Range*	*Description*
There are no parameters. Only the name button will work.			

Inlets

	Name	*Type*	*Range*	*Description*
1	matrix input	Matrix		Input for the video matrix stream.

Outlets

	Name	*Type*	*Range*	*Description*
1	matrix output	Matrix		Video matrix output

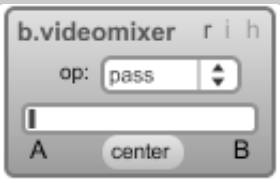
Messages

none

b.videomixer

Mixer for two jitter matrix inputs using *jit.op*. Allows different blending operators.

GUI



	Name	*Type*	*Range*	*Description*
1	op	menu		Mathematical operator
2	Mix A/B	slider	0-127	Mix amount
3	center	button		

Inlets

	Name	*Type*	*Range*	*Description*
1	bang, messages	Bang, symbol		A bang outputs one frame. Input for all the messages.

Outlets

	Name	*Type*	*Range*	*Description*
1	matrix output	Matrix		Video matrix output.
2	Message output	Symbol		Messages from the internal jit.qt.movie object.

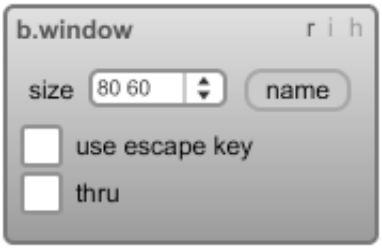
Messages

Format	*Range*	*Description*
bw $i	0/1	Toggles black and white (monochrome) mode
dim $i $i	0-?	Video matrix dimensions.
mirror $i	0/1	Toggles mirror (horizontal flip) mode
start		Starts the currently loaded movie
stop		Stops the currently loaded movie

b.window

Wraper for the *jit.window* object. Creates a *jitter* floating window.

GUI



	Name	*Type*	*Range*	*Description*
1	size	menu		Window size
2	name	toggle		Edit the window name
3	escape key	toggle	0/1	If on, using the escape key will open or close the fullscreen mode
4	thru	toggle	0/1	If on, the input matrix will be sent out the outlet.

Inlets

	Name	*Type*	*Range*	*Description*
1	Matrix input, messages	Matrix, messages		The matrix to be sent to the floating or fullscreen window.

Outlets

	Name	*Type*	*Range*	*Description*
1	matrix output	Matrix		Input matrix output, if *thru* is on.

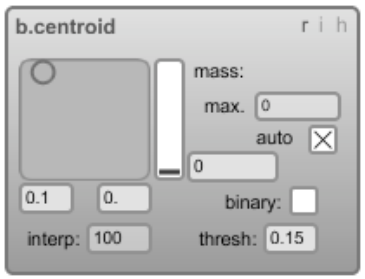
Messages

b.window will receive any other message to the jit.window object, so all the messages of this object can be used.

b.centroid

Based on the *cv.jit.centroids* object, from the *cv.jit* library by *Jean-Marc Pelletier*, b.centroid calculates the horizontal and vertical centroid of the input image, and also the mass.

GUI



	Name	*Type*	*Range*	*Description*
1	center	pictslider		Centroid horizontal and vertical value
2	horizontal	float	0. - 1.	Horizontal value
3	vertical	float	0. - 1.	Vertical value
4	Interp.	int	0 – inf.	Interpolation for the centroid coordinates
5	mass	slider	0 – inf.	Mass value (slider)
6	max.	int		Mass maximum value
7	auto	toggle		Automatically find the mass maximum value
8	mass value	int		Mass value (int)
9	binary	toggle		Use a black and white (binary) image instead of greyscale
10	thresh.	float		Threshold for the binary image

Inlets

	Name	*Type*	*Range*	*Description*
1	Matrix input, messages	matrix, symbol		Image to be analysed. Input for all the messages.

Outlets

	Name	*Type*	*Range*	*Description*
1	matrix output	Matrix		Video matrix output.
2	Centroid coordinates (x y)	Float list	0. – 1.	The centroid horizontal and vertical coordinates, in a *float* list.
3	Centroid coordinates (x y)	Int list	0 – n	The centroid horizontal and vertical coordinates, in a *int* list.
4	Current mass	int		Mass value
5	Mass peak	int		Mass peak value

Messages

Format	*Range*	*Description*
binary $i	0/1	Toggles binary (black and white) mode
dim $i $i	0-?	Video matrix dimensions.
mirror $i	0/1	Toggles mirror (horizontal flip) mode
start		Starts the currently loaded movie
stop		Stops the currently loaded movie

Módulos de Processamento de Som

b.brickwall

An FFT brickwall filter. Only the frequencies inside the defined *bin* range pass thru.
Bins correspond to fft frequency bands.

GUI

	Name	*Type*	*Range*	*Description*
1	high	int	0-255	High *bin* (frequency band) number
2	low	int	0-255	Low *bin* (frequency band) number

Inlets

	Name	*Type*	*Range*	*Description*
1	Signal input	audio	-	Audio signal to be filtered.
2	Low bin number/ messages	Int/ symbol	0-255 -	Low bin number
3	High bin number	Int	0-255	High bin number

Outlets

	Name	*Type*	*Range*	*Description*
1	Signal output	Audio	-	Filtered signal output

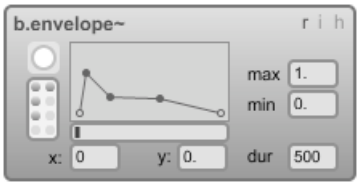
Messages

Format	*Range*	*Description*
high $i	Int	High bin number
low $i $i	Int	Low bin number

b.envelope

A general purpose envelope generator. Can be used for control or audio.

GUI



	Name	*Type*	*Range*	*Description*
1	trigger	bang	-	Trigger the envelope
2	Preset	Preset	0-8	Envelope presets
3	envelope	Function	-	Envelope curve
4	Time slider	Slider		Current time
5	Time (x) value	Int	0-?	Current time
6	Value (y)	float	?-?	Current value
7	Max	Float	-	Maximum value
8	Min	float	-	Minimum value
9	dur	Int	0-?	Envelope duration

Inlets

	Name	*Type*	*Range*	*Description*
1	bang, messages	Bang, symbol		A bang triggers the envelope. Input for all the messages.

Outlets

	Name	*Type*	*Range*	*Description*
1	Y value (float)	float	-	Y value (float)
2	Y value (signal)	signal	-	Y value (signal)
3	Envelope	List	-	Output in line format

Messages

Format	*Range*	*Description*
dur $i	0-?	Envelope duration (miliseconds)
min $i	0-?	Minimum envelope value
max $i	0/1	Maximum envelope value
preset $i	-	Envelope preset number

b.freqsplit

An audio splitter. Splits the input signal in up to 4 frequency regions.

GUI

	Name	*Type*	*Range*	*Description*
1	Split points	Radio	1-3	Number of split points
2	Splitpoint 1	Int	0-255	*Bin* number at splitpoint 1
3	Splitpoint 2	Int	0-255	*Bin* number at splitpoint 2
4	Splitpoint 3	int	0-255	*Bin* number at splitpoint 3

Inlets

	Name	*Type*	*Range*	*Description*
1	Audio input	signal	-	Audio signal to be processed
2	Splitpoint 1/ messages	Int/ Symbol	0-255	*Bin* number at splitpoint 1
3	Splitpoint 2	int	0-255	*Bin* number at splitpoint 2
4	Splitpoint 3	Int	0-255	*Bin* number at splitpoint 3

Outlets

	Name	*Type*	*Range*	*Description*
1	Output 1	signal	-	Frequencies below splitpoint 1
2	Output 2	signal	-	Frequencies between splitpoints 1 and 2
3	Output 3	signal	-	Frequencies between splitpoints 2 and 3
4	Output 4	signal	-	Frequencies above splitpoint 2

Messages

Format	*Range*	*Description*
splitpoint1 $i	0-255	*Bin* number at splitpoint 1
splitpoint1 $i	0-255	*Bin* number at splitpoint 2
splitpoint1 $i	0-255	*Bin* number at splitpoint 3