43-Traitement du signal avec *pfft ~*

Travailler dans le domaine fréquentiel

La plupart des traitements numériques de signaux audio se font dans le domaine temporel. Comme le montrent les autres didacticiels MSP, bon nombre des processus les plus courants de manipulation de l'audio consistent à faire varier des échantillons (ou groupes d'échantillons) en amplitude (modulation en anneau, mise en forme d'onde, distorsion) ou en temps (filtres et délais). La transformation de Fourier rapide (FFT) vous permet de traduire les données audio du domaine temporel au domaine fréquentiel, où vous pouvez directement manipuler le spectre d'un son (les fréquences composantes d'une tranche d'audio).

Comme nous l'avons vu dans le *didacticiel d'analyse 3*, les objets MSP *fft ~* et *ifft ~* vous permettent de transformer des signaux dans et hors du domaine fréquentiel. L'objet *fft ~* prend un groupe d'échantillons (communément appelé *frame*) et les transforme en paires de nombres réels et imaginaires qui contiennent des informations sur l'amplitude et la phase d'autant de fréquences qu'il y a d'échantillons dans la *frame*. Ces nombres sont généralement appelées ***bins*** ou ***bins de fréquence***. (Nous verrons plus loin que les nombres réels et imaginaires ne sont pas eux-mêmes l'amplitude et la phase, mais que l'amplitude et la phase peuvent être dérivées d'eux.) L'objet *ifft ~* effectue l'opération inverse, en prenant des trames d'échantillons du domaine fréquentiel, et en les reconvertissant en un signal audio du domaine temporel que vous pouvez écouter ou traiter ultérieurement. Le nombre d'échantillons dans la trame est appelée ***taille de la FFT*** (ou parfois ***taille du point FFT***). Elle doit être une puissance de 2 telle que 512, 1024 ou 2048 (pour donner quelques valeurs couramment utilisées).

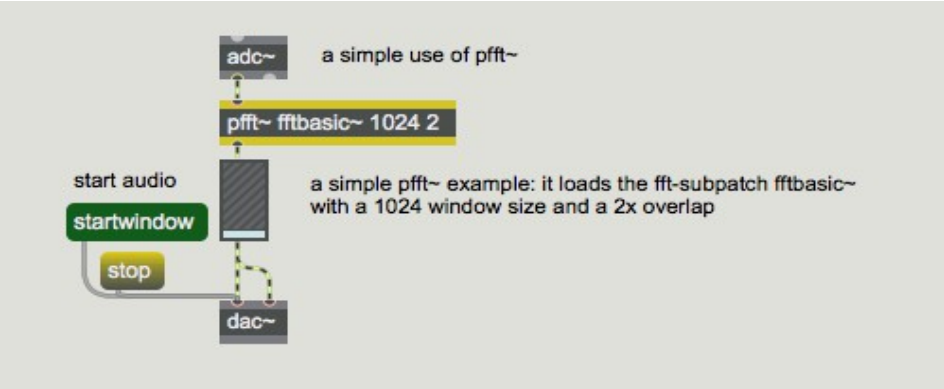
Nous avons également vu que les objets *fft ~* et *ifft ~* fonctionnent sur des trames successives d'échantillons sans faire de chevauchement ou de fondu enchaîné entre elles. Pour les utilisations pratiques de ces objets, nous devons également construire un tel système de chevauchement et de fondu enchaîné autour d'eux, comme indiqué à la fin du didacticiel 3. Il existe plusieurs raisons pour lesquelles il est nécessaire de créer un tel système. En analyse FFT, il y a toujours un compromis entre la résolution en fréquence et la résolution temporelle. Par exemple, si la taille de votre FFT est de 2 048 échantillons, l'analyse FFT vous donne 2 048 bins de fréquence équidistants de 0 Hz. à la fréquence d'échantillonnage (seuls 1024 de ces bins sont utiles; voir le didacticiel 3 pour plus de détails). Cependant, la synchronisation précise des événements qui se produisent dans ces 2048 échantillons sera perdu dans l'analyse, puisque tous les changements temporels sont regroupés dans une seule trame FFT. De plus, si vous modifiez les données spectrales après l'analyse FFT et avant la resynthèse IFFT, vous ne pouvez plus garantir que le signal de domaine temporel produit par l'IFFT correspondra dans les trames successives. Si les vecteurs temporels de sortie ne correspondent pas, vous obtiendrez des clics dans votre signal de sortie. En utilisant une ***fonction de fenêtrage***, vous pouvez compenser ces artefacts en faisant en sorte que les images successives se croisent au fur et à mesure qu'elles se chevauchent. Bien que cela ne compense pas la perte de résolution temporelle, le chevauchement des données d'analyse permet d'éliminer les clics et les pops qui se produisent sur les bords d'une trame IFFT après la resynthèse.

Cependant, cette approche peut souvent être un défi à programmer, et il y a aussi la difficulté de généraliser le patch pour de multiples combinaisons de taille de FFT et de chevauchement. Comme les arguments de *fft ~* / *ifft ~* pour la taille de trame FFT et le chevauchement ne peuvent pas être modifiés, de multiples de chaque sub-patch doivent être créées pour différentes situations. Par exemple, un son percussif nécessitera une analyse avec au moins quatre chevauchements, alors qu'un son raisonnablement statique et riche en harmoniques szmandera une taille de FFT très importante.

Une introduction à l'objet *pfft ~*

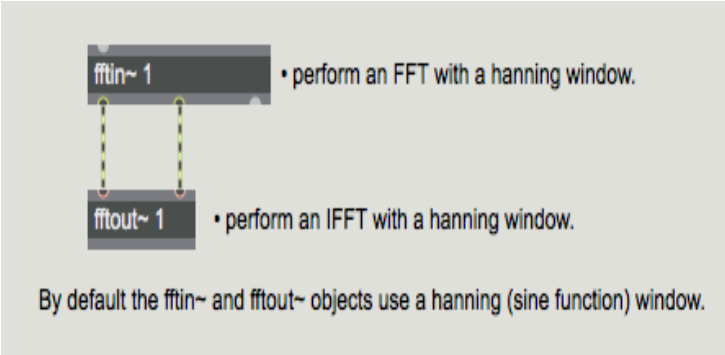
L'objet *pfft ~* répond à de nombreuses lacunes des objets de base *fft ~* et *ifft ~*, en vous permettant de créer et de charger des 'sub-patchs spectraux' spéciaux qui manipulent les données de signal dans le domaine fréquentiel indépendamment du fenêtrage, du chevauchement et de la taille de la FFT. Un seul sub-patch peut donc convenir à plusieurs applications. De plus, l'objet *pfft ~* gère le chevauchement des trames FFT, s'occupe des fonctions de fenêtrage pour vous et élimine les données redondantes en miroir dans le spectre, ce qui le rend à la fois plus pratique à utiliser et plus efficace que les objets traditionnels *fft ~* et *ifft ~*.

L'objet *pfft~* prend comme argument le nom d'un sub-patch spécialement conçu contenant les objets *fftin ~* et *fftout ~* (dont il sera question ci-dessous), un nombre pour la taille de la FFT dans les échantillons et un nombre pour le facteur de chevauchement (ceux-ci doivent être des entiers qui sont une puissance de 2):



Une utilisation simple de *pfft ~*.

Le sub-patch *pff ~,* **fftbasic~** référencé ci-dessus pourrait ressembler à quelque chose comme ceci:



Le sub-patch *fftbasic ~* utilisé dans l'exemple précédent

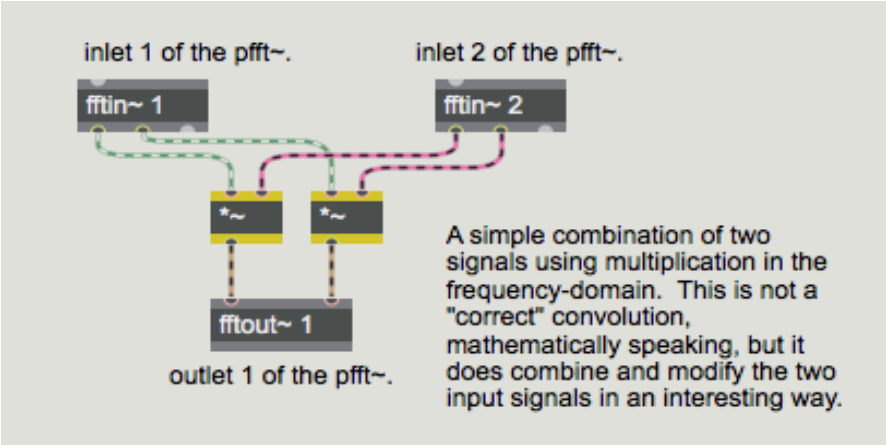
Le sub-patch **fftbasic** ~ présenté ci-dessus prend un signal en entrée, effectue une FFT sur ce signal avec une fenêtre de Hanning (voir ci-dessous) et effectue une IFFT sur le signal FFT, également avec une fenêtre de Hanning. L'objet *pfft ~* communique avec son sub-patch en utilisant des objets spéciaux pour les entrées et les sorties. L'objet *fftin ~* reçoit un signal dans le domaine temporel de son patch parent et le transforme via une FFT dans le domaine fréquentiel. Ce signal dans le domaine temporel a déjà été converti, par l'objet *pfft ~*, en une séquence de trames qui se chevauchent dans le temps, et le signal que *fftin ~* envoie dans le sub-patch spectral représente le spectre de chacune de ces trames entrantes.

Détail technique: par défaut, la taille du vecteur de signal à l'intérieur du sub-patch spectral est égale à la moitié de la taille de la FFT spécifiée en tant qu'argument de la *pfft ~*. Voici pourquoi: pour des raisons d'efficacité, *fftin ~* et *fftout ~* effectuent ce que l'on appelle une FFT réelle, qui est plus rapide que la FFT complexe traditionnelle utilisée par *fft~* et *ifft~*. Ceci est possible car les signaux temporels que nous transformons n'ont pas de partie imaginaire (ou du moins, ils ont une partie imaginaire égale à zéro). Une FFT réelle est une astuce mathématique intelligente qui réorganise l'entrée du domaine temporel uniquement réel de la FFT en parties réelles et imaginaires d'une FFT complexe qui a la moitié de la taille de notre FFT réelle. Le résultat de cette FFT est ensuite réorganisé en un spectre complexe représentant la moitié (de 0Hz à la moitié de la fréquence d'échantillonnage) de notre signal réel d'origine. La taille plus petite de la FFT signifie qu'elle est plus efficace pour le processeur de notre ordinateur, et, comme une FFT complexe produit un spectre en miroir dont seulement la moitié est vraiment utile, la FFT réelle contient toutes les données dont nous avons besoin pour définir et ensuite manipuler le spectre du signal. (Néanmoins, nous pouvons, si nous le souhaitons, définir un argument optionnel à l'objet *pfft~* pour lui demander d'exécuter une ***FFT complexe*** produisant un spectre complet allant de 0Hz à la fréquence d'échantillonnage. Dans ce cas, la taille du vecteur signal du sub-patch spectral est la même que celle de la FFT. Cette méthode est plus coûteuse en termes de calcul, non seulement à cause de la FFT plus grande, mais aussi parce que le sub-patch spectral devra traiter deux fois plus d'échantillons pour la même quantité de données.)

L'objet *fftout~* fait l'inverse, acceptant les signaux du domaine fréquentiel, les reconvertissant en un signal du domaine temporel et en les transmettant via une sortie au patch parent. Les deux objets prennent un argument de nombre (pour spécifier le numéro d'entrée ou de sortie) et un symbole spécifiant la fonction de fenêtre à utiliser. Les fonctions de fenêtre disponibles sont Hanning (par défaut si aucune n'est spécifiée), Hamming, Blackman, Triangle et Square. De plus, le symbole peut être le nom d'un objet *buffer ~* qui contient une fonction de fenêtrage personnalisée. Les différentes fonctions de fenêtrage ont des largeurs de bande et des profondeurs de bande d'arrêt différentes pour chaque bin de la FFT. Une bonne référence sur l'analyse FFT vous aidera à sélectionner une fenêtre en fonction du son que vous essayez d'analyser et de ce que vous voulez en faire. Nous vous recommandons ***le didacticiel d'informatique musicale*** de Curtis Roads ou le ***Guide du scientifique et de l'ingénieur sur le traitement des signaux numériques*** de Steven W. Smith. Généralement, à des fins musicales, la fenêtre de Hanning par défaut fonctionne le mieux, car elle fournit une enveloppe propre sans d'artefacts de modulation d'amplitude en sortie.

Il existe également un argument pratique de fenêtre **nofft**, à appliquer à *fftin ~* et *fftout ~*, qui permet aux trames temporelles superposées de et vers le *pfft ~* d'être passées directement de et vers le sub-patch sans appliquer une fonction de fenêtre ni effectuer une transformée de Fourier. Dans ce cas (parce que la taille du vecteur de signal du sub-patch spectral est la moitié de la taille de la FFT), le signal du domaine temporel est divisé entre les sorties réelles et imaginaires des objets *fftin ~* et *fftout ~*, ce qui peut être assez peu pratique lorsqu'on utilise un chevauchement de 4 ou plus. Bien que l'option **nofft** puisse être utilisée pour envoyer les données du signal de contrôle du patch parent dans le sub-patch spectral, elle n'est pas recommandée pour la plupart des utilisations pratiques de *pfft ~*.

Un sub-patch *pfft ~* plus compliqué pourrait ressembler à ceci:



Un type simple de convolution spectrale

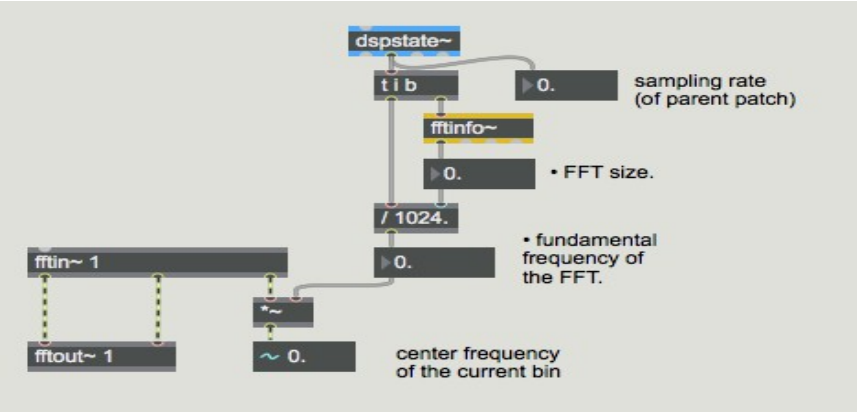
Ce sub-patch prend deux entrées de signaux (qui apparaîtraient comme des 'entrées dans l'objet parent *pfft ~*), les convertit dans le domaine fréquentiel, multiplie les signaux réels entre eux et multiplie les signaux imaginaires entre eux et envoie le résultat à un objet *fftout ~* qui convertit les données du domaine fréquentiel en un signal du domaine temporel. La multiplication dans le domaine des fréquences est appelée ***convolution,*** et c'est la procédure de traitement de signal de base utilisée dans la synthèse croisée (morphing d'un son dans un autre). Le résultat de cet algorithme est que les fréquences des deux analyses qui ont des valeurs d'amplitude plus grandes se renforcent mutuellement, tandis que les fréquences ayant des valeurs d'amplitude plus faibles dans une analyse diminuent ou annulent la valeur de l'autre, qu'elle soit forte ou faible. Le contenu en fréquences que les deux signaux entrants partagent sera conservé, tandis que le contenu de fréquences qui existe dans un signal et pas dans l'autre sera atténué ou éliminé. Cependant, cet exemple n'est pas une "vraie" convolution, car la multiplication des nombres complexes (voir ci-dessous) n'est pas aussi simple que la multiplication effectuée dans cet exemple. Nous verrons plus tard dans ce tutoriel comment réaliser un patch de convolution «correct».

Vous avez probablement déjà remarqué qu'il y a toujours deux signaux à connecter lors de la connexion de *fftin ~* et de *fftout ~*, ainsi que lors du traitement des spectres entre les deux. Ceci est dû au fait que l'algorithme FFT produit des ***nombres complexes*** - des nombres qui contiennent une partie réelle et une partie imaginaire. La partie réelle est envoyée à la sortie la plus à gauche de *fftin ~*, et la partie imaginaire est envoyée par sa deuxième sortie. Les deux entrées de *fftout~* correspondent également aux parties réelles et imaginaires, respectivement. La façon la plus simple de comprendre les nombres complexes est de les considérer comme représentant un point sur un plan à 2 dimensions, où la partie réelle représente l'axe des X (distance horizontale à partir de zéro) et la partie imaginaire représente l'axe des Y (distance verticale à partir de zéro). Nous en apprendrons davantage sur ce que nous pouvons faire avec les parties réelles et imaginaires des nombres complexes plus tard dans ce didacticiel.

La troisième sortie

L'objet *fftin~* possède une troisième sortie qui émet un flux d'échantillons correspondant à l'indice du bin de fréquence actuel dont les données sont envoyées par les deux premières sorties (ceci est analogue à la troisième sortie des objets *fft ~* et *ifft ~* abordés dans le Tutoriel 4). Pour *fftin ~*, cette sortie sort un nombre compris entre 0 et la moitié de la taille de la FFT moins 1. Vous pouvez convertir ces valeurs en valeurs de fréquence (représentant la fréquence "centrale" de chaque bin) en multipliant le signal (appelé signal de synchronisation) par la fréquence de base, ou fondamentale, de la FFT. La fondamentale de la FFT est la fréquence la plus basse que la FFT peut analyser et elle est inversement proportionnelle à la taille de la FFT (c'est-à-dire que des FFT plus

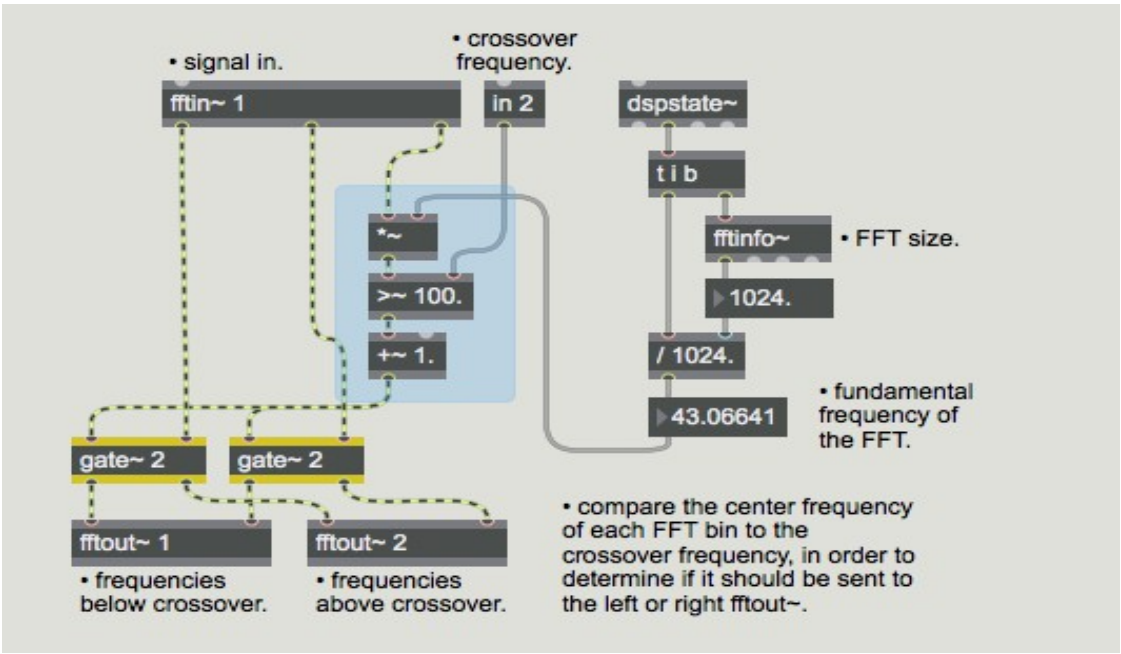
grandes produisent des fréquences de base plus basses encore). La fondamentale exacte de la FFT peut être obtenue en divisant la taille de la trame FFT par la fréquence d'échantillonnage. L'objet *fftinfo* ~, lorsqu'il est placé dans un sub-patch *pfft* ~, vous donnera la taille de trame FFT, la taille de la demi-trame FFT (c.-à-d. Le nombre de bins réellement utilisés dans le sub-patch *pfft* ~) et la taille du saut FFT (le nombre d'échantillons de chevauchement entre les trames fenêtrés). Vous pouvez l'utiliser en conjonction avec l'objet *dspstate* ~ ou l'objet *adstatus* avec l'argument **sr** (fréquence d'échantillonnage) pour obtenir la fréquence de base de la FFT:



Recherche de la fréquence centrale du bin d'analyse en cours.

Notez que dans l'exemple ci-dessus, l'objet *number~* est utilisé uniquement à des fins de démonstration dans ce tutoriel. Lorsque le DSP est activé, le nombre affiché dans la boîte de *nombre* du signal ne semblera pas changer car la boîte de *nombre* du signal affiche par défaut le premier échantillon du vecteur signal, qui dans ce cas sera toujours 0. Pour voir les valeurs de fréquence centrale , vous devrez utiliser l'objet *capture* ~ ou enregistrer ce signal dans un *buffer* ~.

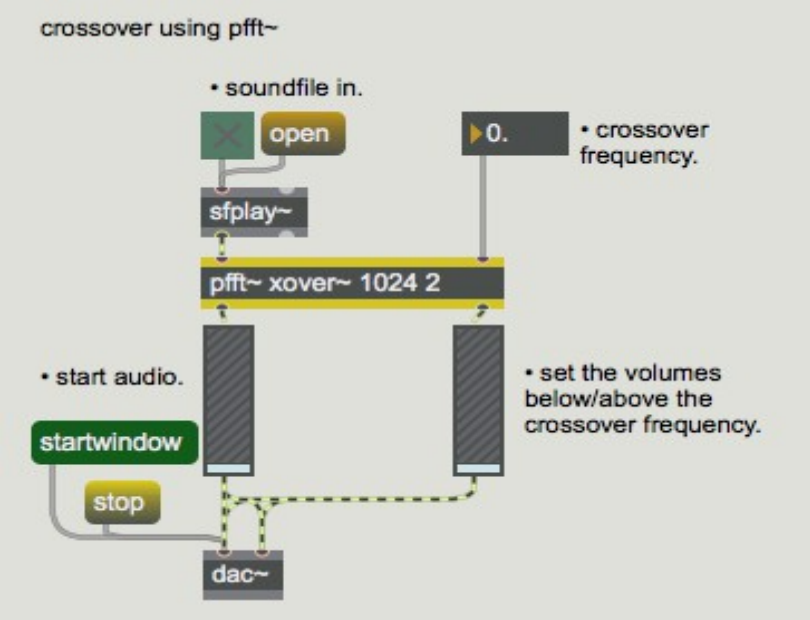
Une fois que vous connaissez la fréquence des bins sortants de *fftin* ~, vous pouvez effectuer des opérations sur les données FFT basées sur la fréquence. Par exemple:



Un simple croisement spectral.

Le sub-patch *pfft~* ci-dessus, appelé **xover~**, prend un signal d'entrée et envoie les données d'analyse à l'un des deux objets *fftout ~* en fonction d'une fréquence de croisement. La fréquence de croisement est envoyée au sub-patch *pfft ~* en utilisant l'objet in, qui transmet les messages Max du patch parent via l'entrée droite de l'objet *pfft ~*. La fréquence centrale du bin actuel - déterminée par la sortie sync en conjonction avec *fftinfo~* et *dspstate~* comme nous l'avons mentionné plus haut - est comparée à la fréquence de croisement.

Le résultat de cette comparaison fait basculer une *gate~* qui envoie les données FFT vers l'un des deux objets *fftout~*: la partie du spectre dont la hauteur est inférieure à la fréquence de recouvrement est envoyée par la sortie gauche du *pfft ~* et la partie dont la hauteur est supérieure à la fréquence de recouvrement est envoyé par la sortie droite. Voici comment ce sub-patch pourrait être utilisé avec *pfft ~* dans un patch:



Une façon d'utiliser le sub-patch **xover ~**

Notez que nous pouvons envoyer des entiers, des flottants et tout autre message Max vers et depuis un sub-patch chargé par *pfft~* en utilisant les objets in et out. (Voir *Tutoriel MSP Polyphonie 1*. Les objets *poly ~* et *out ~* ne fonctionnent pas à l'intérieur d'un *pfft ~*.)

Travailler avec l'amplitude et la phase

Comme nous l'avons déjà appris, les deux premières sorties de *fftin ~* émettent un flux de nombres réels et imaginaires pour la réponse du bin pour chaque échantillon de l'analyse FFT (de même, *fftout ~* attend ces nombres). Il ne s'agit pas de l'amplitude et de la phase de chaque bin, mais plutôt des paires de coordonnées cartésiennes, où x est la partie réelle et y la partie imaginaire, représentant des points sur un plan à 2 dimensions.

L'amplitude et la phase de chaque bin de fréquence sont les coordonnées polaires de ces points, où la distance par rapport à l'origine est l'amplitude du bin et l'angle autour de l'origine est la phase du bin:

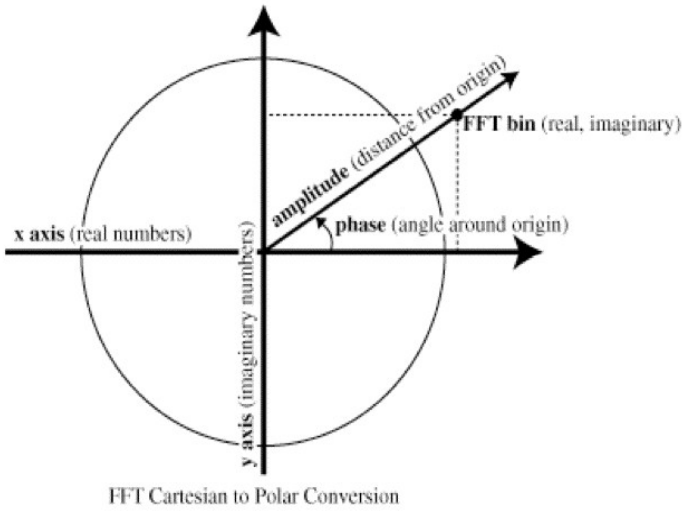
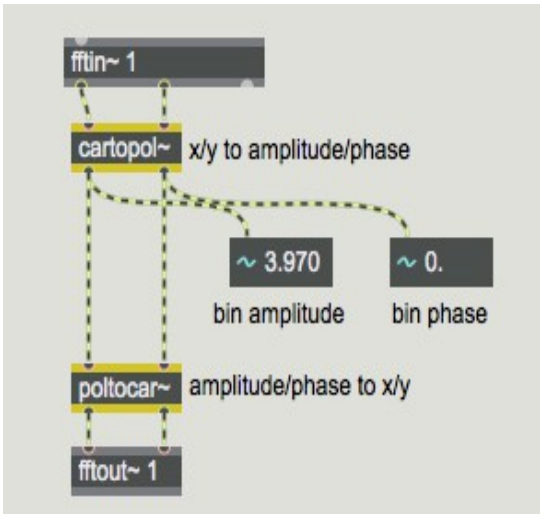


Diagramme unité-cercle montrant la relation entre les valeurs réelles et imaginaires de la FFT, l'amplitude et la phase

Vous pouvez facilement convertir les paires réelles/imaginaires et les paires amplitude/phase en utilisant les objets *cartopol* ~ et *poltocar* ~:



Conversion cartésienne en polaire

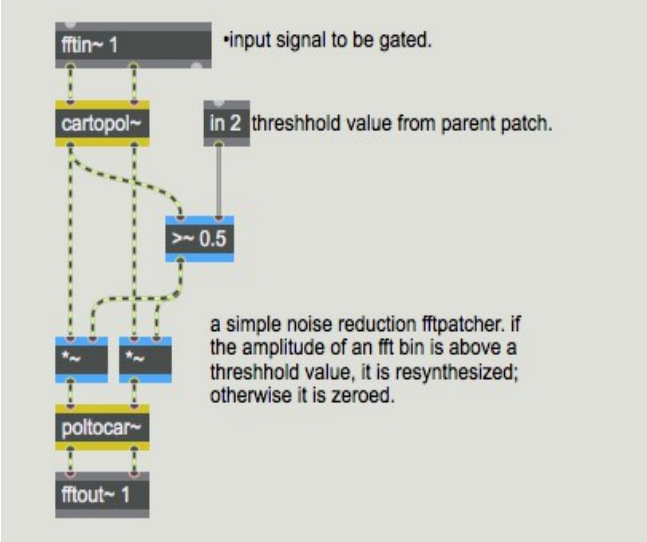
Détail technique: les valeurs d'amplitude émises par la sortie gauche de *cartopol* ~ dépendent naturellement de l'amplitude du signal que vous envoyez à l'objet *pfft* ~. Néanmoins, la valeur d'amplitude maximale pour un signal constant de 1,0 sera égale à la somme des valeurs d'amplitude de la fonction de fenêtrage utilisée. Pour une fenêtre de Hanning, cette valeur est égale à la moitié de la taille de la FFT, et pour une fenêtre carrée, elle est égale à la taille de la FFT. Pour une oscillation, telle qu'une onde sinusoïdale, dont l'amplitude maximale est de 1.0, les valeurs maximales obtenues à partir de la FFT sont égales à un quart de la taille de la fenêtre. Les bins de la FFT situés aux extrémité basses et hautes du spectre peuvent présenter des valeurs maximales légèrement inférieures, et un décalage en courant continu dans le son peut augmenter légèrement les valeurs d'amplitude maximale de l'analyse FFT. Néanmoins, voici une liste de multiplicateurs pour diverses formes de fenêtres afin de trouver la valeur d'amplitude maximale possible d'une onde sinusoïdale à plein volume:

Hanning: Taille FFT * 0.25
Hamming: Taille FFT * 0.27174
Blackman: Taille FFT * 0.21
Triangle: Taille FFT * 0.2495
Square: Taille FFT * 0.5

Ainsi, par exemple, en utilisant une FFT de 512 points avec la fenêtre de Haning par défaut, une onde sinusoïdale de plein volume à la moitié de la fréquence de nyquist aura une valeur de 128 dans le 128ème bin de fréquence (512 * 0.25). Le même scénario utilisant une fenêtre de Blackman nous fournira une valeur de 107,52 dans le 128ème bin de fréquence (512 * 0,21). Même s'il s'agit des valeurs maximales théoriques avec une seule onde sinusoïdale en entrée, les signaux audio du monde réel présenteront généralement des valeurs nettement inférieures, car l'énergie dans les formes d'onde complexes est répartie sur de nombreuses fréquences.

Les valeurs de phase sorties par la sortie droite de *cartopol ~* seront toujours comprises entre -π et π.

Vous pouvez utiliser ces informations pour créer des routines de traitement du signal basées sur des données d'amplitude/phase. Un noise-gate spectral ressemblerait à ceci:



Un noise-gate spectral

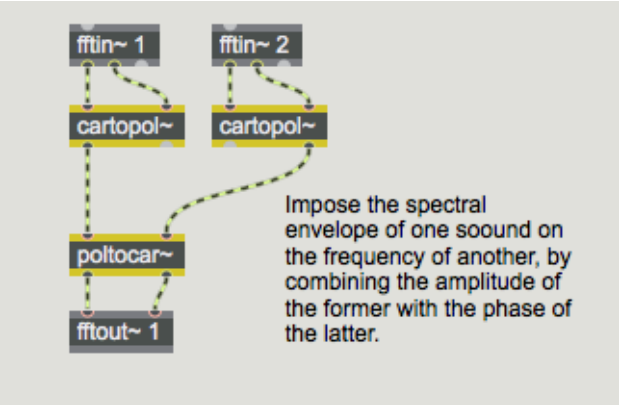
En comparant la sortie d'amplitude de *cartopol ~* avec le signal de seuil envoyé dans l'entrée 2 du *pfft ~*, chaque bin est soit passé soit mis à zéro par les objets * ~. De cette façon, seuls les bins de fréquence qui dépassent une certaine amplitude sont retenus dans la resynthèse (pour des informations sur les valeurs d'amplitude à l'intérieur d'un sub-patch spectral, voir la note technique ci-dessus.).

Convolution et synthèse croisée

Les effets de convolution et de synthèse croisée utilisent généralement des données d'amplitude et de phase pour leur traitement. L'un des effets de synthèse croisée les plus basiques que l'on peut réaliser utilise le spectre d'amplitude d'un son avec le spectre de phase d'un autre. Le spectre de phase étant lié aux informations sur le contenu en fréquence du son, ce type de synthèse croisée peut nous donner le contenu harmonique d'un son "joué" par l'enveloppe spectrale d'un autre son.

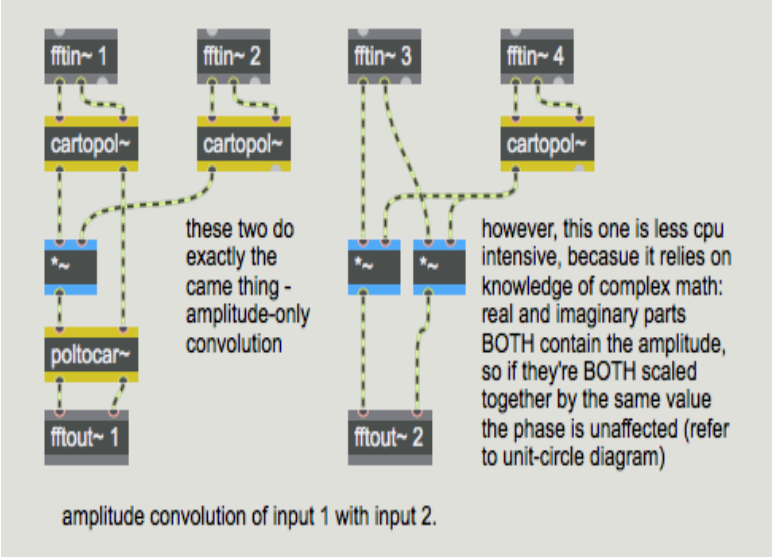
Naturellement, le succès de ce type d'effet dépend fortement du choix des deux sons utilisés.

Voici un exemple de sub-patch spectral qui utilise *cartopol ~* et *poltocar ~* pour réaliser ce type de synthèse croisée:



Synthèse croisée simple

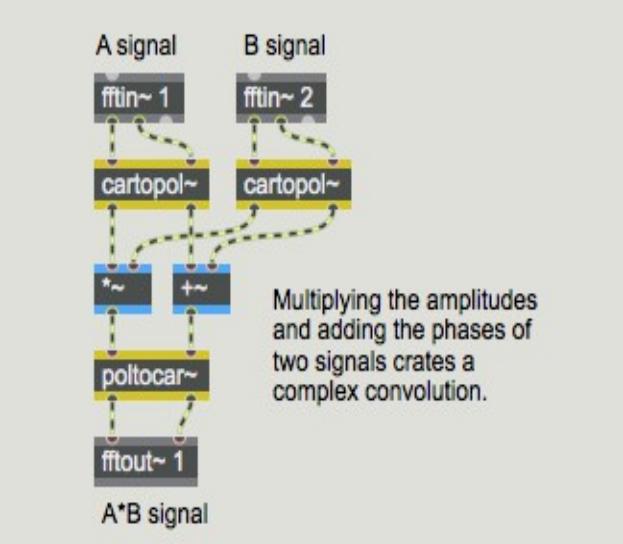
L'exemple de sub-patch suivant montre deux manières de convolutionner l'amplitude d'une entrée avec l'amplitude d'une autre:



Convolution d'amplitude seulement

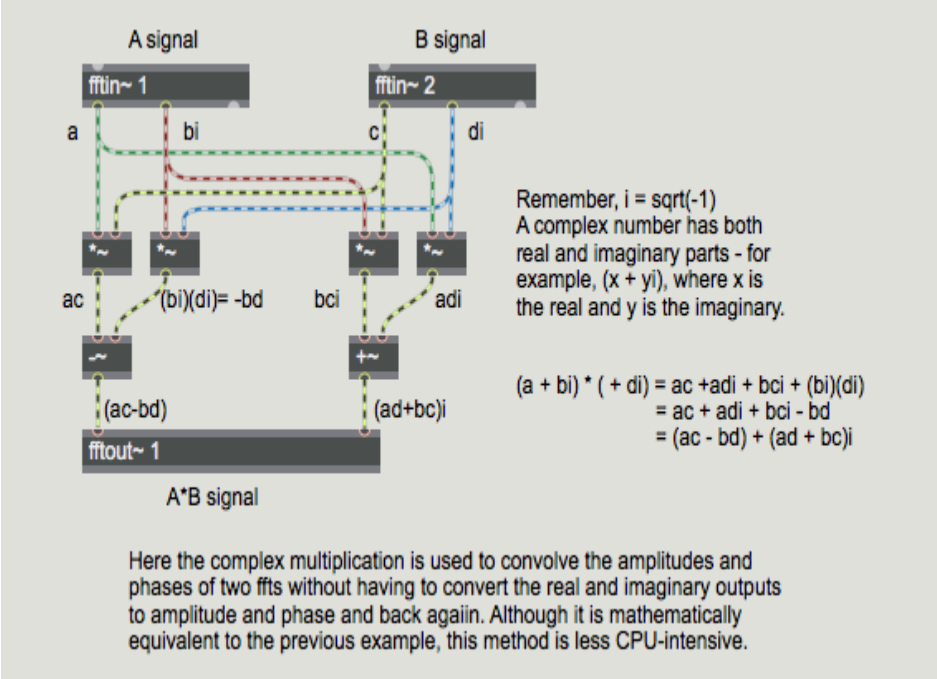
Vous pouvez facilement voir sur le côté gauche de ce sub-patch que les valeurs d'amplitude des signaux d'entrée sont multipliées ensemble. Cela renforce les amplitudes qui sont proéminentes dans les deux sons tout en atténuant celles qui ne le sont pas. La réponse en phase du premier signal n'est pas affectée par la multiplication complexe*real; la réponse en phase du second signal d'entrée est ignorée. Vous remarquerez également que la partie droite du sub-patch est mathématiquement équivalent à la partie gauche, même si elle n'utilise qu'un seul objet *cartopol ~*.

Au début de ce tutoriel, nous avons vu un exemple de la multiplication de deux signaux réels / imaginaires pour effectuer une convolution. Cet exemple est resté simple pour les besoins de l'explication, mais il était, en fait, incorrect. Si vous vous demandez ce qu'implique une multiplication «correcte» de deux nombres complexes, voici une façon de procéder:



La méthode correcte pour faire une convolution complexe

Voici une seconde approche un peu plus astucieuse, pour atteindre le même objectif:



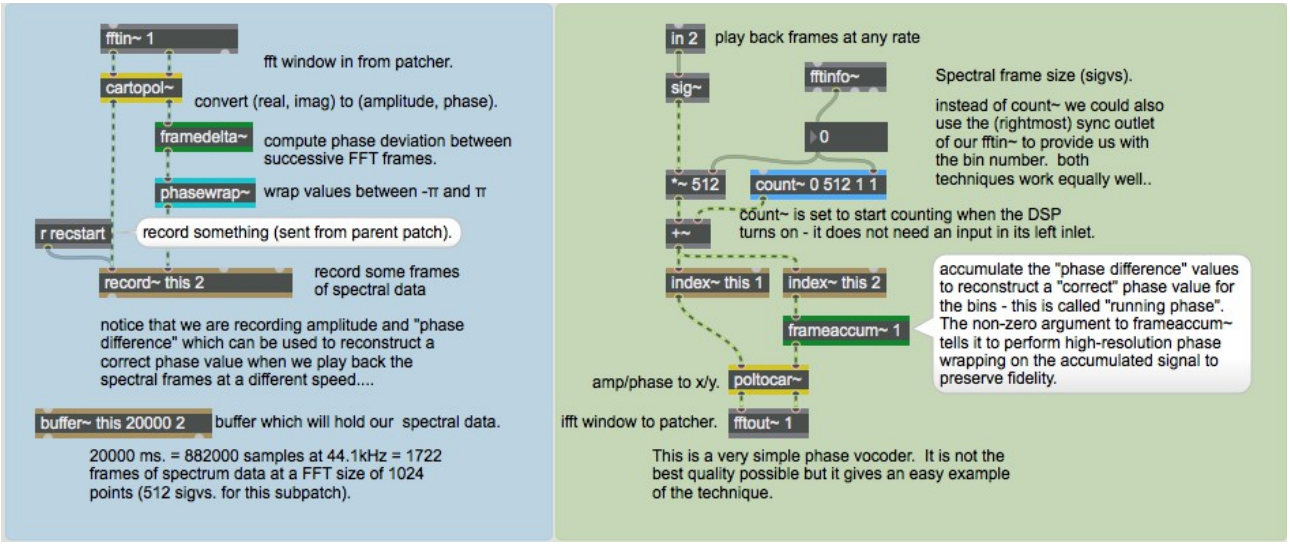
Une manière correcte et intelligente de faire une convolution complexe

Étirement du temps

Les sub-patchs créés pour être utilisés avec *pfft* ~ peuvent utiliser toute la gamme complète d'objets MSP, y compris les objets qui accèdent aux données stockées dans un objet *buffer* ~. (Bien que certains objets qui ont été conçus pour traiter des problèmes de temporisation peuvent ne pas toujours se comporter comme initialement prévu lorsqu'ils sont utilisés à l'intérieur d'un *pfft* ~.)

L'exemple suivant enregistre des données d'analyse spectrale dans deux canaux d'un *buffer* ~ stéréo puis vous permet ensuite de resynthétiser l'enregistrement à une vitesse différente sans modifier sa hauteur d'origine. C’est ce qu’on appelle l’étirement temporel (ou compression temporelle lorsque

le son est accéléré) et fait partie des utilisations importantes du STFT depuis les années 1970.



Enregistrement et lecture dans un sub-patch *pfft ~*

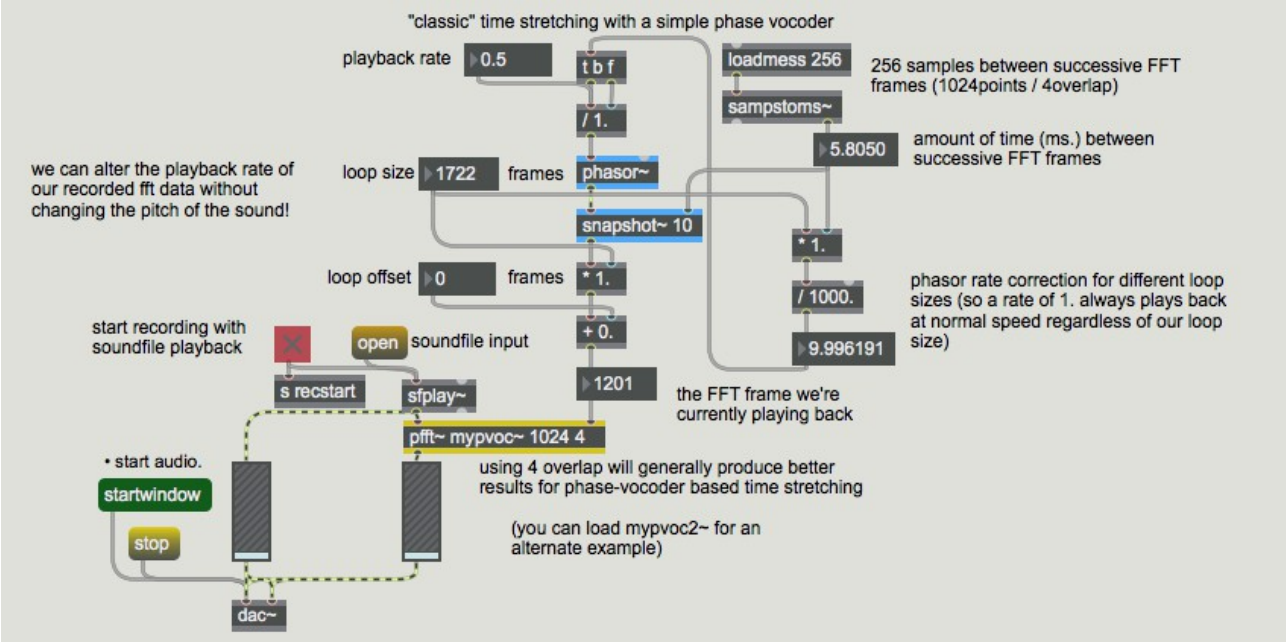
L'exemple de sub-patch enregistre les données spectrales dans un *buffer ~* à gauche et lit les données de ce *buffer ~* à droite. Dans la partie enregistrement du sub-patch, vous remarquerez que nous ne nous contentons pas nous n'enregistrons pas d'enregistrer l'amplitude et la phase en sortie de *cartopol ~*, mais que nous utilisons l'objet *framedelta~* pour calculer la différence de phase (parfois appelée déviation de phase ou dérivée de phase). La différence de phase est tout simplement la différence de phase entre des emplacements de bins équivalents dans des trames FFT successives. La sortie de *framedelta~* est ensuite introduite dans un objet *phasewrap~* pour s'assurer que les données sont correctement contraintes entre -π et π. Les messages peuvent être envoyés à l'objet *record~* depuis le patch parent via l'objet *send* afin de démarrer et d'arrêter l'enregistrement et d'activer la mise en boucle.

Dans la partie lecture du sub-patch, nous utilisons une entrée sans signal pour spécifier le numéro de trame pour la resynthèse. Ce nombre est multiplié par la taille de la trame spectrale et ajouté à la sortie d'un objet *count ~* qui compte de 0 à la taille de la trame spectrale moins 1 afin de pouvoir rappeler successivement chaque bin de fréquence dans la trame donnée en utilisant *index ~* pour lire les deux canaux de notre *buffer~*. (Nous aurions également pu utiliser la sortie *sync* de l'objet *fftin~* à la place de *count ~*, mais nous utilisons la méthode actuelle afin de séparer visuellement les parties enregistrement et lecture de notre sub-patch, ainsi que pour donner un exemple d'utilisation de *count ~* dans le contexte d'un sub-patch spectral.) Vous remarquerez que nous reconstruisons la phase à l'aide de l'objet *frameaccum ~*, qui accumule une valeur de 'phase courante' en effectuant l'inverse de *framedelta ~*. Cette opération est nécessaire car il se peut que nous ne lisons pas les trames d'analyse successivement à la vitesse originale où elles ont été enregistrées. Les signaux sont ensuite reconvertis en valeurs réelles et imaginaires pour *fftout ~* par l'objet *poltocar ~*.

Ceci est un exemple simple de ce que l'on appelle un vocodeur de phase. Les vocodeurs de phase vous permettent d'étirer et de comprimer les signaux indépendamment de leur hauteur en manipulant des données FFT plutôt que des segments du domaine temporel. Si vous considérez chaque trame d'une analyse FFT comme une image unique dans un film, vous pouvez facilement voir comment le fait de parcourir les trames individuelles à des vitesses différentes peut modifier la vitesse apparente à laquelle les choses se produisent. C'est plus ou moins ce que fait un vocodeur de phase.

Notez que, du fait que *pfft ~* fait du recouvrement de fenêtre, la quantité de données qui peut être stockée dans le *buffer ~* dépend des paramètres de l'objet *pfft ~*. Cela peut rendre le réglage correct de la taille du *buffer ~* plutôt délicat, surtout que la taille de la trame spectrale (c'est-à-dire la taille du vecteur signal) à l'intérieur d'un *pfft ~* est la moitié de la taille de la FFT indiquée comme second argument, et que le sub-patch spectral traite des échantillons à une fréquence différente de son patch parent ! Si nous créons un *buffer~* stéréo avec 1000 millisecondes de mémoire d'échantillons, nous aurons 44100 échantillons disponibles pour nos données d'analyse. Si la taille de notre FFT est 1024, chaque trame spectrale occupera 512 échantillons de la mémoire de notre *buffer~* , ce qui représente 86 trames de données d'analyse (44100/512 = 86.13). Ces 86 trames ne représentent cependant pas une seconde de son ! Si nous utilisons un chevauchement de 4 fois, nous traitons une trame spectrale tous les 256 échantillons, donc 86 trames représentent environ 22050 échantillons, soit une demi-seconde de temps par rapport au patch parent. Comme vous pouvez le constater, tout cela peut devenir assez compliqué …

Jetons un coup d'œil au patch parent pour le sub-patch de vocodeur de phase ci-dessus (appelé **mypvoc ~**):



Wrapper pour mypvoc

Remarquez que nous utilisons un objet *phasor ~* avec un objet *snapshot ~* afin de générer une rampe spécifiant l'emplacement de la lecture dans notre sub-patch. Nous pourrions également utiliser un objet *line ~*, voire un curseur, si nous voulions «scruter» nos images d'analyse à la main. Notre patch principal nous permet de modifier la vitesse de lecture d'une boucle de nos données d'analyse. Nous pouvons également spécifier la taille de la boucle et un décalage dans notre collection d'images d'analyse afin de boucler une section donnée des données d'analyse à une vitesse de lecture donnée. Vous remarquerez que la modification de la vitesse de lecture n'affecte pas la hauteur du son, mais uniquement sa vitesse. Vous remarquerez également qu'à des vitesses de lecture très lentes, certaines parties de votre son (généralement les attaques de notes, les consonnes de la parole ou d'autres sons percussifs) deviennent plutôt «étalées» et acquièrent une qualité sonore artificielle.

Résumé

L'utilisation de *pfft* ~ pour effectuer un traitement du signal dans le domaine spectral est généralement plus facile et visuellement plus claire que l'utilisation des objets traditionnels *fft* ~ et *ifft* ~, et vous permet de concevoir des patchs qui peuvent être utilisés à des tailles de FFT et des chevauchements variables. Il existe une myriade d'applications de *pfft* ~ pour le traitement du signal musical, notamment le filtrage, la synthèse croisée et l'étirement temporel.