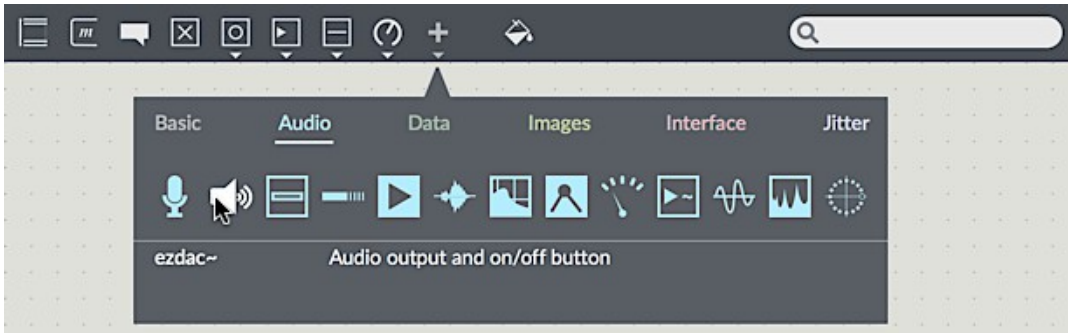
7-Oscillateur à table d'onde

introduction

Dans ce patch de tutoriel, nous allons voir comment utiliser une table d'onde autre qu'une onde sinusoïdale pour générer du son en utilisant l'objet *cycle ~*. En cours de route, nous verrons comment générer des courbes plus complexes à l'aide de l'objet *line ~* et nous aurons un aperçu de la façon dont MSP stocke les données audio dans la mémoire de l'ordinateur à l'aide de l'objet *buffer ~*.

Interrupteur audio on/off: *ezdac ~*

Dans ce patch de tutoriel, l'objet *dac ~* utilisé dans les exemples précédents a été remplacé par un bouton avec une icône de haut-parleur. Il s'agit de l'objet *ezdac ~*, un objet d'interface utilisateur disponible dans la barre d'outils de la *Top Patcher Window Toolbar*.



ezdac ~ est un bouton d'activation / désactivation de l'audio, disponible dans la barre d'outils de la fenêtre Top Patcher

L'*ezdac ~* fonctionne beaucoup comme le *dac ~*, sauf qu'en cliquant dessus l'audio est activé ou désactivé. Il peut également répondre aux messages de démarrage et d'arrêt dans son entrée gauche, comme le *dac ~*. Contrairement à *dac ~*, toutefois, il est câblé pour la sortie des canaux 1 et 2 sur votre périphérique audio, ce qui peut ne pas le rendre approprié pour travailler avec un son multicanal. Pour nos tutoriels, cependant, cela devrait être suffisant. Le bouton *ezdac ~* est mis en surbrillance lorsque le son est activé.

Un son stocké: *buffer ~*

Dans les exemples précédents, l'objet *cycle ~* a été utilisé pour lire de manière répétée 16K de valeurs décrivant un cycle d'une onde cosinusoïdale. En fait, un objet *cycle ~* peut lire n'importe quel ensemble de valeurs, en les traitant comme un seul cycle d'une forme d'onde. Ces nombres doivent être stockés dans la mémoire de votre ordinateur et peuvent être chargés à partir d'un fichier son, générés par programme par un algorithme, ou même dessinés à la main. L'objet MSP qui conserve ces morceaux de mémoire s'appelle *buffer ~*.

L' objet *buffer ~* peut être utilisé facilement; il ne requiert qu'un seul argument: un nom pour l'identifier de façon unique. Toute donnée audio stockée dans ce *buffer ~* devient associée à ce nom. Le nom du *buffer ~* est arbitraire et ne doit pas nécessairement correspondre au nom du fichier audio qui y est chargé; la seule exigence est qu'il ne soit pas en conflit avec d'autres éléments nommés dans les patchs Max ouverts (tels que les objets *table* ou les paires *send / receive*). Comme vous pouvez le constater, nous avons nommé le *buffer ~* ***richtone***.

Les messages envoyés à l'objet vous permettent de charger des fichiers audio, de redimensionner ou

de vider la mémoire du *buffer* ~. L'astuce est la suivante: l'objet *buffer* ~ lui-même ***ne lit aucun fichier audio***; il se contente ed conserver les données d'échantillonnage et leur associe un nom que les autres objets MSP peuvent utiliser pour accéder aux données. On peut faire en sorte qu'un objet *cycle* ~ lise ce *buffer* ~ en tapant le même nom comme argument. La valeur de la fréquence initiale de l'objet *cycle* ~, juste avant le nom du tampon, est optionnelle.

Pour obtenir le son dans *buffer* ~, envoyez-lui un message **replace** ou **read**, suivi du nom d'un fichier audio dans le chemin de recherche du programme Max. Si vous n'indiquez pas le nom d'un fichier, Max ouvrira une boîte de dialogue Ouvrir un document, vous permettant de sélectionner un fichier audio à charger. Le message **replace** o u **read** lit un fichier audio et ***dimensionne*** automatiquement la mémoire de l'objet *buffer* ~ en fonction de la durée du fichier audio.

Quelle que soit la durée du son dans l'objet *buffer* ~, l'objet *cycle* ~ utilisera par défaut 512 échantillons de celui-ci pour sa forme d'onde. (Si vous le souhaitez, vous pouvez spécifier un point de départ dans le *buffer* ~ pour que *cycle* ~ commence sa forme d'onde, soit avec un argument supplémentaire à *cycle* ~ soit avec un message **buffer_ofset** à *cycle* ~. Vous pouvez également modifier l'attribut **buffer_sizeinsamps** pour définir la taille de la table d'ondes.) Dans l'exemple de patch, nous utilisons un fichier audio contenant exactement 512 échantillons, appelé **gtr512.aiff**.

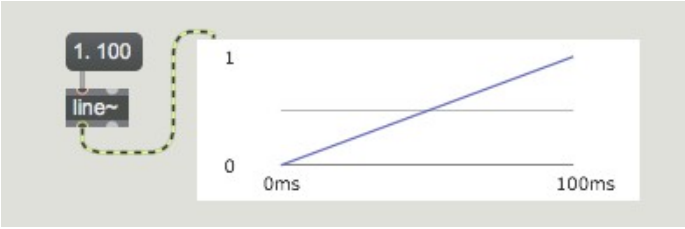
> ***Détail technique**: En **fait**,* cycle ~ *Chris utilise 513 échantillons. Le 513e échantillon est utilisé uniquement que pour l'interpolation à partir du 512e échantillon. Lorsque* cycle ~ *est utilisé pour créer une forme d'onde périodique, comme dans cet exemple de patch, le dernier échantillon doit être le même que le 1er échantillon. Si le* buffer ~ *ne contient que 512 échantillons, comme dans cet exemple,* cycle ~ *fournit un 513e échantillon identique au premier*.

• Dans le tutoriel, cliquez sur la boîte de *message* quit dit lire **gtr512.aiff**. Cela charge le fichier audio dans notre *buffer* ~ (nommé **richtone**). Cliquez ensuite sur l'objet *ezdac* ~ pour activer l'audio. Cliquez sur la boîte de *message* **A**. Vous devriez entendre une brève rafale de ce qui ressemble à du bruit. Cliquez sur le *message* intitulé **B**. Vous devriez entendre un son riche et oscillant qui s'atténue et qui s'estompe pendant une seconde. Cliquez tour à tour sur les autres boîtes de *message*, et écoutez le résultat. Notez les différentes valeurs dans les listes envoyées aux objets *line* ~, ainsi que les différentes fréquences sur lesquelles sont réglées les objets *cycle* ~ dans le didacticiel.

Plusieurs autres objets peuvent utiliser les données d'un *buffer* ~, comme vous le verrez dans les chapitres suivants.

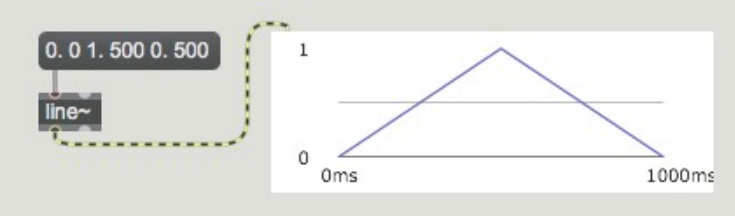
Créer des enveloppes complexes avec *line* ~

Dans l'exemple de patch précédent, nous avons utilisé *line* ~ pour créer un signal changeant linéairement en lui envoyant une liste de deux nombres. Le premier nombre dans la liste était une valeur cible et le second était la durée, en millisecondes, nécessaire à *line* ~ pour atteindre la valeur cible.



line ~ reçoit une valeur cible (1.) et une période de temps pour y arriver (100 ms)

Si nous le souhaitons, nous pouvons envoyer à *line* ~ une liste plus longue contenant de nombreuses paires de nombres valeur-temps (jusqu'à 64 paires de nombres). De cette manière, nous pouvons faire en sorte qu'un objet *line* ~ remplisse une fonction plus élaborée composée de nombreux segments de ligne adjacents. Après avoir terminé le premier segment de ligne, *line*~ se dirige immédiatement vers la valeur cible suivante dans la liste, en prenant le temps spécifié pour y arriver. De cette manière, nous pouvons créer de grandes courbes de fonction pour les synthétiseurs, couramment appelés des ***enveloppes***.



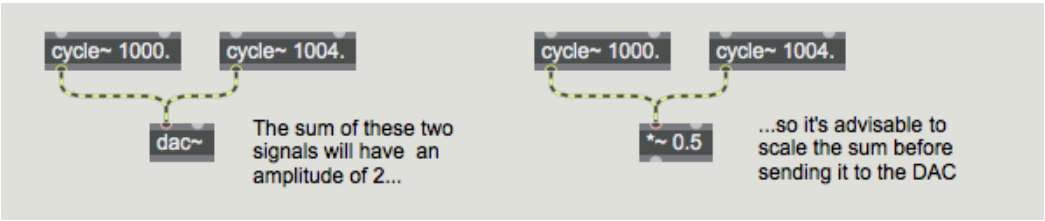
Une fonction composée de segments de ligne - une "enveloppe" classique

Les utilisateurs de synthétiseur connaissent bien l'utilisation de ce type de fonction pour générer des enveloppes telles que les courbes 'ADSR' qui contrôlent l'attaque, la décroissance, le maintien et le relâchement de l'amplitude d'un son de manière indépendante. C'est ce que nous faisons dans cet exemple de patch, bien que nous puissions choisir le nombre de segments de ligne que nous souhaitons utiliser pour l'enveloppe.

• Cliquez à nouveau sur les boîtes de *message* dans l'ordre et déchiffrez comment les listes de chaque objet *line* ~ affectent le son. Notez que chaque boîte de **message** commence par un **0** suivi d'une virgule, ce qui envoie un message individuel **0** à l'objet *line* ~ immédiatement suivi de la liste suivante. L'envoi d'un nombre seul à un objet *line*~ le transforme ***immédiatement*** en une valeur. Une notation équivalente serait de commencer nos listes d'enveloppes avec 0 0 (aller à 0 dans 0 millisecondes, c'est-à-dire dans l'instant).

Ajouter des signaux pour produire un son composite

Chaque fois que deux signaux ou plus sont connectés à la même entrée de signal, ces signaux sont additionnés et leur somme est utilisée par l'objet récepteur.



Plusieurs signaux sont ajoutés (mélangés) dans une entrée de signal

L'addition de signaux numériques est équivalente au mélange de gain unitaire en audio analogique. Il est important de noter que même si tous vos signaux ont une amplitude inférieure ou égale à 1, la somme de ces signaux peut facilement dépasser 1. En mode MSP, il est possible d'avoir un signal d'une amplitude supérieure à 1 n'importe où dans la chaîne du signal, mais avant d'envoyer le signal a u *dac*~, vous devez le mettre à l'échelle (généralement avec un objet *~) pour conserver son amplitude inférieure ou égale à 1. Un signal dont l'amplitude est supérieur à 1 sera déformé par le *dac* ~.

Dans l'exemple de patch, nous utilisons trois objets *cycle* ~ différents qui font osciller la forme d'onde stockée dans le *buffer* ~ nommé **richtone**. Bien que jusqu'à présent, nous les avons tous joués l'un après l'autre, ils pourraient tous être mixés ensemble pour produire un son d'instrument

composite.

• Réglez le volume de notre patch de didacticiel à un niveau maximal de **0,3** pour éviter l'écrêtage (rappelez-vous que nous utilisons trois sons différents, chacun avec une sortie maximale hypothétique de 1). Cliquez sur le *button* en haut du patcheur pour lire les trois signaux simultanément.

Chacune des trois sons a une enveloppe d'amplitude différente, ce qui entraîne l'évolution du timbre de la note au cours de sa durée d'une seconde. Dans le même temps, même si les trois sonorités sont jouées à partir du même échantillon, ils sont réglés sur des fréquences différentes, ce qui crée un spectre beaucoup plus riche que celui du fichier audio d'origine utilisé pour la table d'ondes. Comme nous le verrons dans le prochain tutoriel, le mélange de tables d'ondes de fréquences différentes est une technique clé de ce qu'on appelle la ***synthèse additive***.

Résumé

L'objet *ezdac ~* est un bouton permettant d'activer et de désactiver le son. L'objet *buffer ~* stocke un son dans la mémoire de l'ordinateur. Vous pouvez charger un fichier audio dans le *buffer ~* avec un message **replace**. Si un objet *cycle ~* à un argument saisi qui lui donne le même nom qu'un objet *buffer ~*, le *cycle ~* utilisera 512 échantillons de ce *buffer ~* comme forme d'onde au lieu de l'onde cosinus par défaut.

Chaque fois que vous connectez plus d'un signal à une entrée de signal donnée, l'objet récepteur additionne ces signaux et utilise la somme comme entrée dans cette entrée. Faites attention lorsque vous mélangez (ajoutez) des signaux audio, afin d'éviter toute distorsion causée par l'envoi d'un signal d'amplitude supérieure à 1 au *dac ~*.

L' objet *line ~* peut recevoir une liste dans son entrée gauche, composée en un maximum de 64 paires de nombres représentant des valeurs cibles et des temps de transition. Il produira un signal qui change linéairement d'une valeur cible à une autre dans les délais spécifiés. Cela peut être utilisé pour créer une fonction de segments de ligne décrivant toute forme souhaitée, ce qui est particulièrement utile comme signal de contrôle pour les enveloppes d'amplitude. Vous pouvez réaliser des fondus enchaînés entre les signaux en utilisant différentes enveloppes d'amplitude provenant de différents objets *line ~*.