11-Listes et matrices

Ce tutoriel montre comment utiliser les ***listes*** Max et l'objet *jit.fill* pour remplir tout ou partie d'une matrice, et comment récupérer tout ou partie du contenu d'une matrice sous forme de liste avec *jit.spill*. Nous allons également montrer l'utilisation de ***noms*** de matrice pour accéder au contenu d'une matrice à distance, concept qui sera présenté plus loin dans les ***didacticiels*** 12, 16 et 17.

Noms de matrice

À gauche du patch, vous verrez un objet bleu *jit.matrix*. Le premier argument donne à la matrice un nom spécifique, **smallbox**. Les arguments restants indiquent que la matrice aura **1** plan de données de type ***char*** et que la matrice n'aura qu'une seule dimension avec **12** cellules.

Cette matrice a un nom unique: **smallbox**.

Dans le ***tutoriel 2***, nous avons expliqué que chaque matrice a un nom. Si nous ne donnons pas explicitement un nom à une matrice, Jitter choisira un nom de manière arbitraire (généralement un quelque chose d'étrange comme "u040000114", afin que le nom soit unique). Le nom est utilisé pour désigner l'emplacement dans la mémoire de l'ordinateur où le contenu de la matrice est stocké. Alors, pourquoi donner notre propre nom à une matrice? De cette façon, nous connaissons le nom et nous pouvons facilement indiquer aux autres objets comment trouver le contenu de la matrice. En se référant au nom d'une matrice, les objets peuvent partager les mêmes données et accéder au contenu de la matrice à distance, sans recevoir de message **jit_matrix**.

> L'utilisation par Jitter du nom de la matrice pour faire référence à son emplacement mémoire est analogue au fonctionnement de l'objet *value* de Max. Vous pouvez avoir plusieurs objets *value* portant le même nom. Vous pouvez également stocker une valeur numérique dans n'importe quel d'entre eux et récupérer la même valeur de n'importe quel autre. Mais il n'existe en réalité qu'un seul emplacement mémoire pour ce nom, de sorte qu'ils partagent tous les mêmes données. De la même manière, vous pouvez avoir plus d'un objet *jit.matrix* avec le même nom et ils partageront tous les mêmes données. Certains autres objets, comme *jit.fill*, peuvent accéder au contenu de cette matrice en connaissant simplement son nom.

jit.fill

Dans le ***didacticiel 2***, nous avons montré comment placer une valeur numérique dans un emplacement particulier de la matrice à l'aide du message **setcell**, et comment récupérer le contenu d'un emplacement avec le message **getcell**. Nous allons maintenant montrer comment utiliser l'objet *jit.fill* pour placer une liste entière de valeurs dans une matrice. (Plus tard dans ce chapitre, nous montrerons également comment récupérer plusieurs valeurs à la fois dans une matrice.)

Dans le coin supérieur gauche du patch, ce trouve une boîte de *message* contenant une **liste** de douze valeurs numériques. Elle est attachée à un objet **smallbox** *jit.fill*. L'argument **smallbox** fait référence à un nom de matrice.

jit.fill place une liste de valeurs dans la matrice nommée

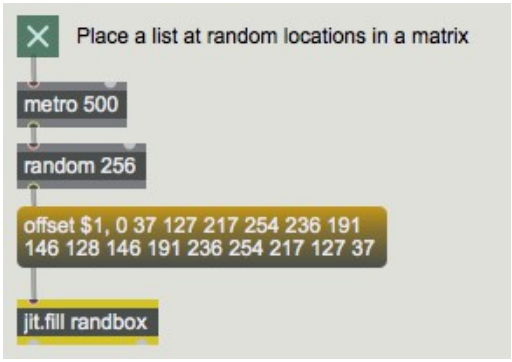
• Cliquez sur la boîte de *message* pour envoyer la liste des valeurs à l'objet *jit.fill.**smallbox***. L'objet *jit.fill.**smallbox*** place ces valeurs dans la matrice nommée "smallbox". Pour vérifier que cela est vrai, cliquez sur le *button* situé au-dessus de l'objet *jit.matrix **smallbox*** afin d'afficher le contenu de la matrice "smallbox". Les valeurs sont imprimées dans la console Max par *jit.print* et affichées sous forme de niveaux de gris dans un *jit.pwindow* étroite.

Dans cet exemple, la liste était exactement de la bonne longueur pour remplir la matrice entière. Cependant, cela n'est pas nécessairement le cas. Nous pouvons placer une liste de n'importe quelle longueur dans n'importe quelle partie contiguë d'une matrice 1D ou 2D.

L'attribut offset

• Ouvrez le sub-patch **random_lists** en double-cliquant dessus.

Par défaut, *jit.fill* place la liste de valeurs au tout début de la matrice. Vous pouvez toutefois diriger la liste vers n'importe quel endroit de la matrice en définissant l'attribut **offset** de *jit.fill*. Le sub-patch **random_lists** illustre l'utilisation de la fonctionnalité **offset**.



Spécifiez d'abord l'offset, puis fournissez la liste

Cet exemple choisit un index de cellule au hasard, utilise ce nombre aléatoire comme argument d'un message **offset** pour l'objet *jit.fill* **randbox**, puis envoie une liste de 16 éléments à stocker à partir de cet index dans la matrice **randbox**.

• Il suffit de cliquer sur le *toggle* pour démarrer le *metro*. Toutes les demi-secondes, la liste de 16 éléments sera écrite dans un nouvel emplacement de la matrice **randbox**.

La liste a été écrite à quatre endroits dans la matrice "graybox"

• Vous pouvez utiliser le message **clear** pour mettre à zéro le contenu de la matrice **randbox**, puis observer comment le *metro* écrit la liste dans de nouveaux emplacements aléatoires. Notez que le *metro* frappe aussi sur la *jit.matrix* **randbox** pour envoyer son contenu à *jit.pwindow* pour la distribution. Lorsque vous avez terminé, éteignez le *metro*.

Utiliser multiSlider

• Ouvrez le sub-patch **draw_list**.

Jusqu'ici, nous avons montré comment mettre une liste prédéterminée de valeurs dans une matrice. Si vous souhaitez générer une telle liste de nombres de manière interactive dans Max et les placer dans une matrice en temps réel, vous devez utiliser un objet Max conçu pour construire des listes. Nous allons examiner deux de ces objets: *multislider* et *zl*.

L'objet *multislider* affiche un ensemble de curseurs individuels et envoie la position de tous ses curseurs à la fois sous forme de liste de valeurs. (Les curseurs peuvent être aussi petits qu'1 pixel de large, ce qui peut le faire ressembler davantage à un graphique qu'à un ensemble de commandes individuelles.) Il envoie la liste complète lorsque vous cliquez dans la fenêtre pour déplacer l'un des curseurs. Et il renvoie la liste lorsque vous relâchez le bouton de la souris. Dans le sub-patch **draw_list**, nous avons configuré un *multislider* pour qu'il contienne 256 curseurs qui envoient des valeurs de 0 à 255, ce qui est parfait pour envoyer une liste de 256 caractères à l'objet *jit.fill* **graybox**.

• Utilisez la souris pour dessiner dans le *multislider*, en réglant ses 256 curseurs. Lorsque vous relâchez le bouton de la souris, la liste de 256 valeurs est envoyée à l'objet *jit.fill* **graybox**. Remarquez comment la luminosité des cellules de la matrice correspond à la hauteur des curseurs.

Dès que *jit.fill* reçoit une liste dans son entrée, il écrit les valeurs dans la matrice nommée (à la position spécifiée par l'attribut **offset**). Dès que cela est fait, *jit.fill* envoie un **bang** par sa sortie gauche. Vous pouvez utiliser ce **bang** pour déclencher une autre action, telle que l'affichage de la matrice.

> Dans les deux premiers exemples, nous avons délibérément évité d'utiliser le **bang** de la sortie gauche de *jit.fill*, afin de préciser que *jit.fill* écrit à distance dans la matrice nommée sans être physiquement connecté à l'objet *jit.matrix*. Le **bang** de la sortie gauche de *jit.fill* est cependant pratique pour déclencher la sortie de la matrice dès qu'elle a été remplie.

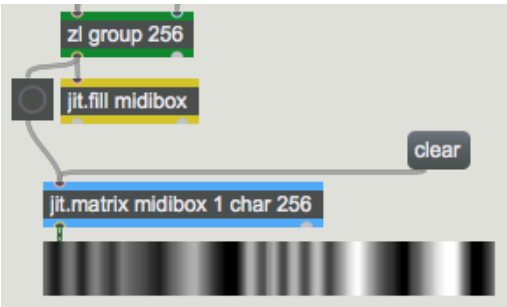
Utiliser zl

• Ouvrez le sub-patch **collect_values**.

Dans certaines situations, vous pouvez utiliser une matrice pour stocker des messages numériques qui se sont produits quelque part dans le patch: des messages MIDI, des nombres provenant d'un objet de l'interface utilisateur, etc. Les messages **setcell** et **getcell** de *jit.matrix* sont utiles pour cela, mais pour une autre façon de faire est de rassembler les messages dans une liste, et de les placer dans la matrice en une seule fois avec *jit.fill*.

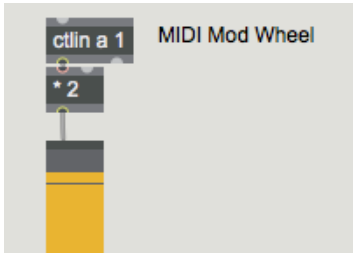
L'objet *zl* est un objet de traitement de liste polyvalent avec de nombreux modes de comportement possibles, en fonction de son premier argument. Lorsque son premier argument est **group**, il collecte les messages reçus dans son entrée gauche jusqu'à ce qu'il en ait accumulé un certain nombre, puis envoie les nombres sous forme de liste unique. (Les valeurs sont groupées dans l'ordre dans lequel elles ont été reçues.) Ainsi, dans le sub-patch **collect_values**, nous avons placé un objet *zl* **group 256** qui collectera 256 valeurs dans son entrée de gauche et, après en avoir reçu 256, il les enverra par sa sortie gauche sous forme de liste (et videra sa propre mémoire).

• Déplacez le *slider* vers le haut et le bas pour générer 256 valeurs d'entrée pour l'objet *zl*. Lorsque *zl* a reçu 256 nombres, il les envoie sous forme de liste à la **midibox** *jit.fill* - qui les écrit dans la matrice **midibox** - puis **bang** sur l'objet *jit.matrix* **midibox 1 *char* 256** pour afficher la matrice.



zl envoie une liste de 256 éléments dans la matrice **midibox**, puis lance *jit.matrix* pour afficher le résultat.

• Si vous avez un contrôleur de clavier MIDI connecté à votre ordinateur, vous pouvez utiliser la molette de modulation du clavier MIDI pour déplacer le *slider*. (L'interaction entre MIDI et Jitter est expliquée en détail dans les chapitres suivants du didacticiel.)



Les valeurs sont doublées pour occuper la plage 0-254, ce qui les rend utiles comme données **char** pour la matrice.

Vous pouvez modifier la longueur de la liste que *zl* collecte en envoyant une nouvelle longueur de liste dans l'entrée droite à partir de la boîte de *nombre* de ***List Lenght***. Et vous pouvez dire où dans la matrice vous voulez le placer, en envoyant un message **offset** à *jit.fill* à partir de la boîte de *nombre* ***Location***. En variant la longueur de la liste et l'emplacement, vous pouvez placer n'importe quel nombre de valeurs dans n'importe quelle région contiguë de la matrice.

• Essayez de modifier la ***List Lenght*** de *zl* (par exemple **100**) et de définir l'emplacement de l'attribut **offset** de *jit.fill* (par exemple **50**), puis déplacez encore le curseur pour placer une liste de valeurs à cet emplacement particulier de la matrice.

jit.fill avec des matrices à plans multiples

• Ouvrez le sub-patch **fill_separate_panes**.

jit.fill fonctionne très bien avec les matrices à plans multiples, mais il ne peut remplir qu'un seul plan à la fois. Le plan auquel *jit.fill* accède est spécifié dans son attribut **plane**. Dans le sub-patch **fill_separate_panes**, nous avons créé une autre matrice, avec quatre plans de données de ***char***, nommée **colorbox**. Nous avons mis en place trois *multisliders* et trois objets *jit.fill*, chacun adressant un plan de couleur différent de la matrice **colorbox**.

Remplir chaque plan indépendamment

• Faites glisser les trois objets *multisliders* colrés pour remplir chacun des trois plans de couleur.

Il s'agit d'un moyen pratique pour générer différentes courbes d'intensité dans les plans RVB d'une matrice. La *jit.pwindow* qui affiche la matrice a en fait une largeur de 256 pixels, de sorte que chacune des 64 cellules de la matrice est affichée sur une bande de 4 pixels de large. Si vous activez l'attribut **interp** de *jit.pwindow*, les différences entre les bandes adjacentes seront lissées par interpolation.

• Cliquez sur le *toggle* situé au-dessus de la boîte de *message* **interp $ 1** pour envoyer le message **interp 1** à *jit.pwindow*. (Notez que cela envoie également un **bang** à *jit.matrix* pour ré-afficher son contenu.

Identique à l'exemple précédent, mais avec interpolation activée dans *jit.pwindow*.

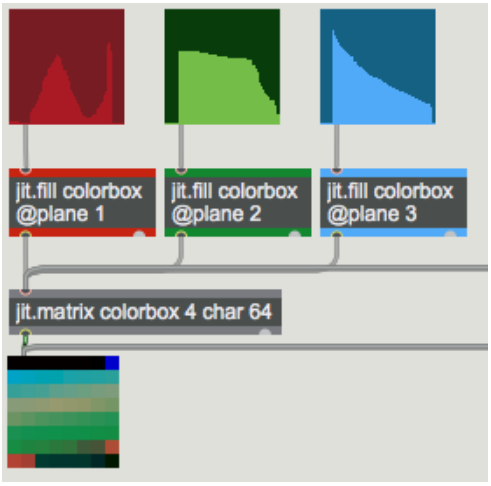
jit.fill avec des matrices 2D

Jusqu'ici, tous nos exemples ont impliqué des matrices unidimensionnelles. Que se passe-t-il lorsque vous utilisez une liste (qui est un tableau unidimensionnel) pour remplir une matrice bidimensionnelle via *jit.fill*? L'objet *jit.fill* utilisera la liste pour remplir autant qu'il le peut dans la première dimension (c'est-à-dire qu'il ira jusqu'à la ligne spécifiée), puis il passera à la ligne suivante et continuera au début de la cette ligne. Nous avons fait en sorte que vous puissiez voir cet effet de recouvrement en action.

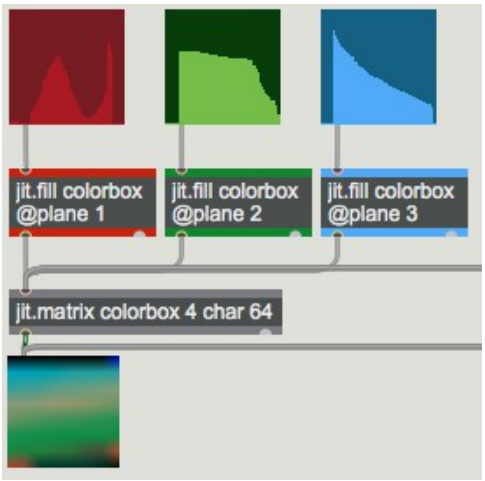
• Cliquez sur le bouton 2D. Cela modifiera l'objet **colorbox** *jit.matrix* afin qu'il contienne une matrice bidimensionnelle 8 x 8, et redimensionnera également la fenêtre *jit.pwindow* à une forme plus appropriée. Chaque fois que vous modifiez les dimensions d'une matrice, celle-ci perd son contenu. Vous devrez donc cliquer à nouveau dans les trois *multisliders* pour remplir à nouveau la matrice. Vous envoyez toujours une **liste** de 64 éléments à chacun des objets *jit.fill*, et ils remplissent chacune des huit lignes de la matrice avec huit éléments.

Important: Bien que nous n'ayons pas démontré l'utilisation de l'attribut **offset** avec une matrice 2D dans ce patch, il est utile de mentionner que lorsque l'attribut **name** de *jit.fill* nomme une matrice 2D, l'attribut **offset** requiert deux arguments: un pour l'offset x et un pour l'offset y.

jit.fill ne fonctionne que pour les matrices 1D et 2D.



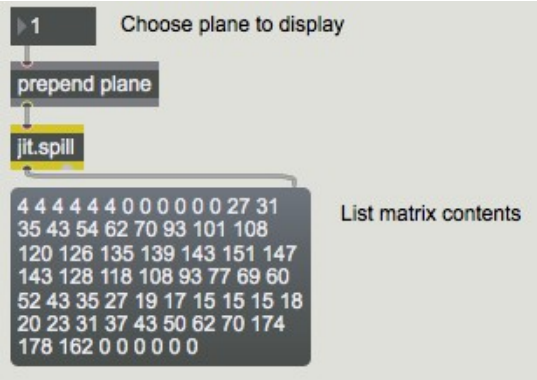
Le même exemple, chaque liste étant encapsulée dans une matrice 8x8 (non interpolée)



Le même exemple, affiché avec interpolation

jit.spill

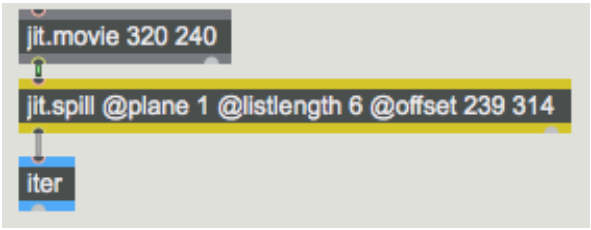
L'objet complémentaire de *jit.fill* est *jit.spill*. Il prend un message **jit_matrix** dans son entrée et envoie les valeurs de la matrice à sa sortie gauche sous forme de **liste** Max. Vous avez peut-être remarqué que pendant que vous utilisiez le *multislider* rouge, l'objet *jit.spill* ci-dessous envoyait les valeurs du plan 1 (rouge) par la sortie gauche et définissait le contenu d'une boîte de *message*.



Le contenu du plan 1 de la matrice "colorbox", affiché sous forme de liste Max

Bien que cela ne soit pas démontré dans le patch, *jit.spill* possède également des attributs **listlength** et **offset** qui vous permettent de spécifier exactement le nombre de valeurs que vous souhaitez lister et à partir de ***quel emplacement*** dans la matrice.

Si vous avez besoin d'avoir les valeurs comme une série immédiate de messages de nombres individuels, plutôt que comme un message de liste unique, vous pouvez envoyer la liste à l'objet Max *iter*.



Obtenir quelques valeurs de la matrice et les transformer en messages séparés

jit.iter

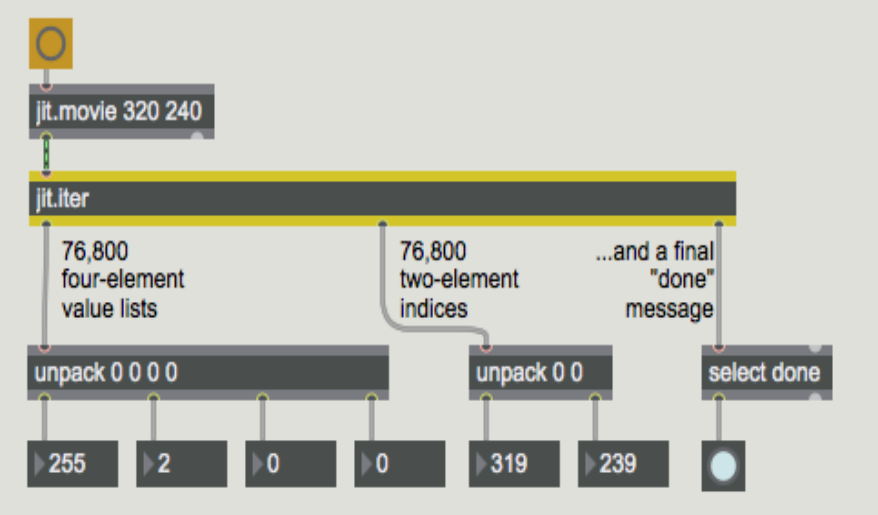
• Ouvrez le sub-patch **individual_values**.

Lorsque vous devez récupérer chaque valeur d'une matrice, il existe un objet appelé *jit.iter*. Lorsqu'il reçoit un message **jit_matrix** dans son entrée, il envoie une séquence de messages aussi rapide que possible: l'index de cellule (par sa sortie centrale) suivi de la ou des valeurs dans cette cellule (par sa sortie gauche), pour chaque cellule de la matrice dans l'ordre. Pour une grande matrice, il peut s'avérer extrêmement compliqué d'envoyer des messages Max en un seul clic du planificateur de Max; donc, quand il a fini de rapporter toutes les valeurs d'une matrice, *jit.iter* envoie un message **done** sur sa sortie droite.

Dans le sub-patch **individual_values**, il y a un objet *jit.iter* qui reçoit les informations de matrice de l'objet *jit.matrix* **readbox 1 char 256**. Nous utilisons un objet *swap* pour changer l'ordre de l'index de la cellule (qui sort de la sortie centrale de *jit.iter*) et la valeur de la cellule (sortant de la sortie gauche de *jit.iter*). Nous utilisons ensuite la valeur de cette cellule comme la valeur y que nous souhaitons stocker dans un objet *table*, et nous utilisons l'index de la cellule comme index de l'axe x de *table*.

• Cliquez sur l'objet *multislider* pour envoyer son contenu à *jit.fill* (qui à son tour enverra un **bang** à l'objet *jit.matrix* et communiquera son contenu à *jit.iter*. Double-cliquez ensuite sur l'objet *table* pour ouvrir sa fenêtre graphique et constater qu'elle contient les mêmes valeurs que la matrice **readbox**.

Notez que cette technique d'utilisation de *jit.iter* pour remplir une table fonctionne bien avec une matrice unidimensionnelle à un plan de taille modeste, car une table est un tableau unidimensionnel. Cependant, la matrice d'un objet *jit.movie*, par exemple, a deux dimensions et quatre plans. Donc, dans ce cas, la sortie de la sortie centrale (index de cellule) de *jit.iter* sera une liste à deux éléments et la sortie de la sortie gauche (valeur) serait une liste à quatre éléments.



Qu'est-ce que tu vas faire avec tous ces chiffres?

Néanmoins, pour les matrices unidimensionnelles, ou les petites matrices 2D, ou même pour rechercher une valeur ou un motif particulier dans une matrice plus grande, *jit.iter* est utile pour balayer une matrice entière.

Sommaire

Pour placer des valeurs individuelles dans une matrice, ou récupérer des valeurs individuelles d'une matrice, vous pouvez utiliser les messages **setcell** et **getcell** dans *jit.matrix* (comme indiqué dans le *didacticiel Jitter 2*). Pour placer toute une liste entière de valeurs dans une matrice, ou récupérer une liste de valeurs d'une matrice, utilisez les objets *jit.fill* et *jit.spill*. Ces objets fonctionnent bien pour s'adresser à n'importe quel plan d'une matrice 1D ou 2D, et ils vous permettent d'adresser n'importe quelle longueur de liste à n'importe quel emplacement de cellule de départ dans la matrice.

Les objets *multislider* et *zl* sont utiles pour créer des messages de liste Max en temps réel. Avec *multislider*, vous pouvez dessiner une liste en faisant glisser les curseurs avec la souris. Avec *zl* **group**, vous pouvez rassembler de nombreuses valeurs numériques individuelles en une seule liste, puis les envoyer toutes à *jit.fill* en une seule fois.

Vous spécifiez l'emplacement de la cellule de départ dans la matrice en définissant l'attribut **offset** de *jit.fill* (ou *jit.spill*). L'objet *jit.fill* nécessite que vous définissiez son attribut **name** (soit en lui envoyant un message [**name**], soit en saisissant un argument [**name**]), en spécifiant le nom de la matrice qu'il va remplir. Il accède à la matrice en utilisant ce nom, et envoie un **bang** par sa sortie chaque fois qu'il écrit une liste dans la matrice. Vous pouvez utiliser ce **bang** pour déclencher d'autres actions. Dans les ***tutoriels 12, 16 et 17***, nous montrons quelques utilisations pratiques de l'accès à une matrice par son nom.

Pour afficher chaque valeur d'une matrice entière, vous pouvez envoyer la matrice à *jit.iter*.