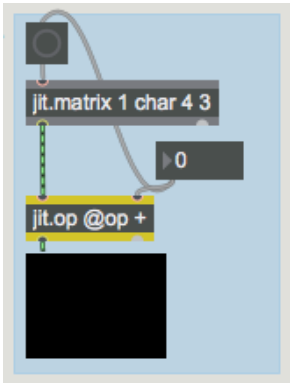
3-Opérations mathématiques sur une matrice

Ce tutoriel montre comment vous pouvez effectuer des opérations mathématiques simples sur les données stockées dans une matrice Jitter. Nous vous montrerons comment utiliser l'objet *jit.op* pour effectuer une mise à l'échelle arithmétique de cellules de la matrice, ou de plans individuels dans ces cellules.

Le patch du didacticiel est divisé en trois exemples simples d'opérations mathématiques que vous pouvez effectuer avec l'objet *jit.op*. L'objet *jit.op* effectue des opérations mathématiques sur des matrices entières de données en une fois plutôt que sur des nombres individuels.



Ajout d'une valeur constante à toutes les cellules d'une matrice.

Le premier exemple montre un objet *jit.matrix* lié à un *jit.op* dont la sortie peut être visualisée par un objet *jit.pwindow*. Chaque fois que vous modifiez la boîte de *nombre* reliée à l'entrée droite de l'objet *jit.op*, un **bang** émettra une nouvelle matrice à partir de l'objet *jit.matrix*. Comme vous pouvez le constater d'après ses arguments, l'objet *jit.matrix* génère une matrice 4x3 de données **char** à plan unique (c'est-à-dire des données dans la plage 0-255). L'objet *jit.pwindow* visualisera cette matrice pour vous comme une image en niveaux de gris. En faisant glisser la boîte de *nombres*, vous modifierez le niveau de gris affiché dans la *jit.pwindow,* du noir (**0**) au blanc (**255**).

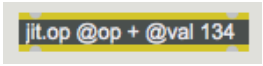
Il est important de réaliser que l'objet *jit.matrix* émet une matrice Jitter dont toutes les cellules sont à 0. Si vous deviez connecter les objets *jit.matrix* et *jit.pwindow* ensemble et contourner l'objet **jit.op**, vous verriez une image noire, peu importe le nombre de fois que vous envoyez un message **bang** à l'objet *jit.matrix*. L'objet *jit.op* ***ajoute*** une valeur (définie par la boîte de *nombre*) à toutes les cellules de la matrice Jitter envoyée entre les objets *jit.matrix* et *jit.op*.

Opération @ -Sign

Nous avons dit plus haut que l'objet *jit.op* ajoute une valeur à toutes les cellules de sa matrice d'entrée. L'objet *jit.op* ajoute une valeur (plutôt que, disons, de diviser ou de multiplier) grâce à l'argument de son attribut **op** (écrit **@op** dans l'objet). L'argument suivant **@op** est un symbole (ou une liste de symboles, comme nous le verrons dans un instant) qui définit les calculs que le *jit.op* effectue sur sa matrice d'entrée. Dans ce cas, nous pouvons voir que l'attribut **op** est fixé à la valeur +, ce qui signifie qu'il effectue une simple addition sur toute matrice qui arrive dans son entrée gauche. La valeur entière dans l'entrée droite est ajoutée à toutes les cellules de la matrice. Cette valeur est qualifiée de ***scalaire***, car elle ajoute la même valeur à l'ensemble de la matrice (dans le didacticiel 9, nous montrons comment *jit.op* peut également effectuer des calculs en utilisant deux matrices Jitter).

Remarque importante: La modification de la valeur scalaire dans l'entrée droite de l'objet *jit.op* ne produit pas une nouvelle matrice. Si vous déconnectiez le cordon de raccordement entre la boîte de *nombres* et l'objet *button*, l'objet *jit.pwindow* ne vous montrera plus rien de nouveau. La raison en est que, comme avec la plupart des objets Max, la plupart des objets Jitter ne sortent des données que lorsque quelque chose entre dans leur entrée la plus à gauche. Dans le cas ci-dessus, chaque fois que vous modifiez la boîte de *nombre*, l'objet *jit.op* stocke la nouvelle valeur scalaire. Dès que cela se produit, l'objet *button* envoie un **bang** à l'objet *jit.matrix*, lui faisant envoyer une nouvelle matrice Jitter (avec toutes ses valeurs à 0) dans l'entrée gauche de l'objet *jit.op*, déclenchant une matrice de sortie que vous pouvez voir. Si vous placez un point d'arrêt sur le cordon de patch au-dessus de *button*, et que vous parcourez l'ordre des messages avec la commande **Step** (shift-cmd-T), vous verrez comment cela se passe. (Consultez le *didacticiel Max 5: Ordre des messages et débogage* pour plus de détails sur la façon de tracer les messages Max avec la fonction watchpoints.)

La valeur scalaire peut également être fournie comme une constante en utilisant l'attribut **val** de **jit.op**. Par exemple, si nous voulions toujours ajouter 134 à toutes les cellules d'une matrice Jitter entrante, nous pourrions utiliser cet objet et supprimer la boîte de *nombre*:

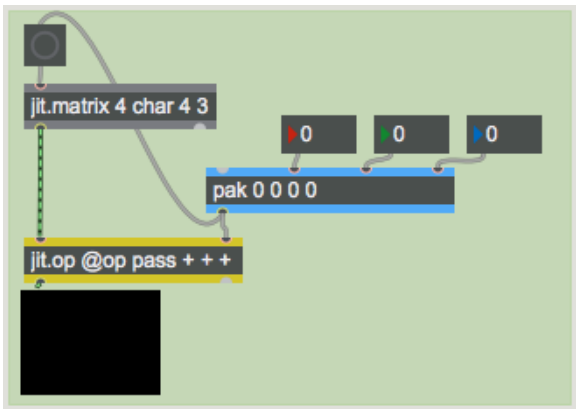


Définir un scalaire en tant qu'attribut.

De même, si nous voulions changer l'opération mathématique effectuée par un objet *jit.op* donné, nous pouvons envoyer le message **op** suivi du symbole mathématique pertinent dans l'entrée gauche de l'objet.

Opérations mathématiques sur plusieurs plans de données

Le deuxième exemple montre une instance plus complexe d'utilisation de *jit.op* pour ajouter des valeurs à une matrice entrante.



Utilisation de scalaires séparés pour chaque plan d'une matrice

Ce patch est similaire au premier, avec la différence importante que nous travaillons maintenant avec une matrice à 4 plans. Ceci est indiqué par le premier argument de l'objet *jit.matrix* qui génère la matrice que nous utilisons. La fenêtre *jit.pwindow* montre maintenant les choses en couleur, interprètant les quatre plans des données de la matrice Jitter comme dese canaux de couleur séparés d'alpha, rouge, vert et bleu. Notre objet *jit.op* dans cet exemple a une liste de quatre symboles pour son attribut **op**: chaque symbole définit l'opération mathématique pour un plan de la matrice entrante. Dans ce patch, nous allons faire passer le premier plan (alpha) sans le modifier, et ajouter des nombres à chacun des autres plans. (Vous pouvez mélanger les opérateurs comme ceci à votre

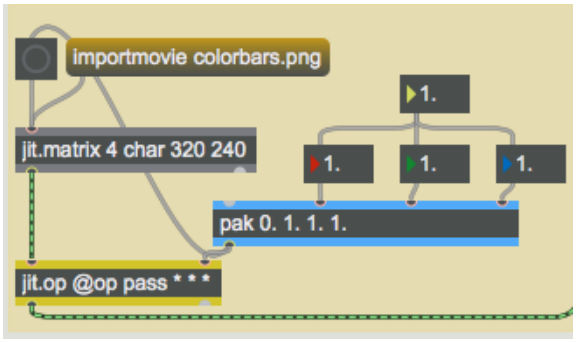
guise).

L'objet *pak* qui alimente l'entrée droite de notre objet *jit.op* prend quatre entiers et les empaquette dans une liste. La seule différence entre *pak* et l'objet Max *pack* est que *pak* produira une nouvelle liste lorsqu'un nombre est modifié (contrairement à l'objet *pack,* qui a besoin d'un nouveau nombre ou d'un **bang** dans l'entrée gauche pour produire une nouvelle liste). Les quatre nombres de la liste générée par *pak* déterminent les scalaires pour chaque plan de la matrice entrant dans l'objet *jit.op.* Dans l'exemple ci-dessus, rien ne sera ajouté dans le plan 0 (le premier argument de l'attribut op est pass). Les plans 1, 2 et 3 se verront ajouter 161, 26 et 254 respectivement. Notre objet *jit.pwindow* interprétera les cellules de la matrice de sortie comme de belles nuances de magenta (même si nous ne voyons qu'une seule couleur, il y a en fait 12 cellules différentes dans la matrice, toutes définies avec les mêmes valeurs).

> **Remarque importante**: si nous avions décidé de n'utiliser qu'une seule valeur pour l'attribut **op** de l'objet *jit.op* ci-dessus (et de n'utiliser qu'un seul nombre comme scalaire), *jit.op* utiliserait cet opérateur mathématique et cette valeur scalaire pour tous les ***plans*** de la matrice entrante.

Modification des couleurs d'une image

Le troisième exemple montre une utilisation de *jit.op* sur une matrice contenant déjà des données pertinentes:



Multiplier des plans individuels avec des scalaires

• Cliquez sur la boîte de *message* **importmovie colorbars.pict**. Le message **importmovie** à *jit.matrix* charge une seule image d'un fichier image ou d'un fichier vidéo dans la matrice Jitter stockée par l'objet. Il met à l'échelle l'image sur le disque aux dimensions de sa propre matrice (dans ce cas, 320 par 240).

En cliquant sur l'objet *button*, vous affichez les barres de couleur de calibrage de l'image dans la fenêtre *jit.pwindow* à droite du patch. Dans ce cas, notre objet *jit.op* a ses opérateurs arithmétiques réglés sur **pass** pour le plan alpha et * (multiplier) pour les autres plans. Comme nous travaillons avec une image à 4 plans, nous définissons chacun des scalaires en utilisant une liste de 4 nombres à virgule flottante. Les valeurs de 1. dans les plans 1 à 3 vous montreront l'image telle qu'elle apparaît à l'origine:

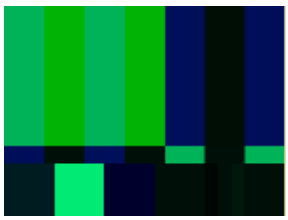
Tous les scalaires à 1.

Si vous définissez les scalaires sur 1., 0. et 0., vous devriez voir l'image suivante:

De méchants rouges.

Tous les plans (sauf le plan 1) de la matrice contenant les barres de couleur ont été multipliés par 0. Cela éliminera les plans alpha, vert et bleu de la matrice, ne laissant que le rouge (plan 1).

Si vous définissez des valeurs intermédiaires (telles que 0., 0., 1. et 0.5) en tant que scalaires pour *jit.op*, vous obtiendrez une image dont les barres de couleur seront différentes:



Joli, n'est-ce pas?

Dans ce cas, le canal alpha est ignoré et le canal rouge est mis à zéro. Les valeurs du plan bleu correspondent toutes à la moitié de ce qu'elles étaient. Le canal vert (plan 2) reste inchangé.

> **Remarque importante**: certains scalaires mathématiques de *jit.op* sont exprimés en nombres à virgule flottante et certains sont exprimés comme des entiers. Cela dépend de l'opérateur pertinent (défini par l'attribut **op**), ainsi que du **type** de la matrice d'entrée. Étant donné que tous les exemples de ce didacticiel utilisent des matrices de type ***char***, il est logique d'utiliser des nombres entiers lors de l'ajout à ces matrices (les nombres à virgule flottante seront tronqués, car les données de la matrice doivent rester sous forme d'entiers compris entre 0 et 255). Si nous utilisions une matrice **float32** comme entrée, il serait parfaitement logique d'y ajouter des nombres à virgule flottante. De même, il est raisonnable de multiplier une matrice **char** par un scalaire à virgule flottante (240 * 0.5 = 120, un entier). Cependant, puisque la matrice produite par *jit.op* sera toujours une matrice **char** (voir la remarque ci-dessous), vous n'obtiendrez que des valeurs comprises entre 0 et 255.

Si vous expérimentez avec les valeurs scalaires, vous verrez que vous pouvez facilement faire disparaître certaines barres de couleurs ou les fusionner avec les barres voisines. Cela est dû au fait que les barres de couleur sont toutes définies sur des valeurs de couleur standard avec des plages similaires. Si vous n'affichez qu'un seul canal à la fois (en réglant tous les plans sauf un sur 0), quatre des sept barres situées en haut de l'écran afficheront de la couleur.

Nous avons montré les opérateurs + et * dans ce didacticiel, mais en fait, l'objet *jit.op* peut effectuer un grand nombre d'autres opérations mathématiques. Pour une liste complète des opérateurs possibles, reportez-vous à la page de référence, ou double-cliquez sur le sub-patch **p op_list** dans le fichier d'aide *jit.op*.

Dimensionnement

Lorsque vous créez un objet *jit.pwindow*, il apparaît dans la console Max avec une largeur de 80 pixels sur une hauteur de 60 pixels. Vous pouvez modifier sa taille à l'aide de sa ***boîte de croissance***, comme de nombreux objets d'interface utilisateur dans Max. Si vous souhaitez modifier sa taille avec précision, vous pouvez le faire en utilisant son inspecteur ou en lui envoyant le message **size** suivi d'une largeur et d'une hauteur, en pixels:

Changer la taille d'un *jit.pwindow*

Si vous envoyez à un objet *jit.pwindow* d'une certaine taille (en pixels) une matrice d'une taille différente (en cellules), l'objet *jit.pwindow* mettra à l'échelle la matrice entrante pour vous montrer la matrice entière. Si vous envoyez une très petite matrice à une très grande *jit.pwindow*, vous verrez de la pixellisation (régions rectangulaires dans l'image où la couleur reste exactement la même). Si vous envoyez une petite fenêtre *jit.pwindow* à une grande matrice, vous risquez de perdre des détails à divers degrés dans ce que vous voyez.

> **Remarque importante**: dans l'exemple ci-dessus, notre *jit.matrix* contenant les barres de couleur avait une taille (spécifiée par sa liste de **dim**) de 320 par 240 cellules, le **planecount** de 4 et un type de **char**. L'objet *jit.op* (et la plupart des objets Jitter que vous rencontrerez) reconnaît cette information et s'adapte pour effectuer son calcul sur l'ensemble de la matrice et générer une matrice avec les mêmes spécifications. Si nous devions changer l'objet *jit.matrix* à une taille différente, celui-ci reconnaîtrait instantanément le changement et se réadapterait. L'objet *jit.pwindow* s'adapte également à la matrice entrante, mais d'une manière légèrement différente. Si la matrice entrante est plus petite que ses propres dimensions, il utilise des données en double pour remplir tous ses pixels. (Cela entraîne l'effet de pixelisation décrit dans le paragraphe précédent.) Si la matrice entrante est plus grande que ses propres dimensions, il sera obligée d'ignorer certaines des données et n'affichera que ce qu'il peut. Ainsi, même si les objets *jit.pwindow* du patch de Tutorial ne correspondent jamais à la taille (en cellules) de leur matrice d'entrée, ils font de leur mieux pour s'adapter à la taille de la matrice de l'objet *jit.op*. La *jit.pwindow* du dernier exeemple vous montre autant qu'elle le peut la matrice entière produite par l'objet *jit.op*, mais elle doit ignorer une ligne et une colonne sur deux afin de faire tenir la matrice 320x240 qu'elle reçoit dans sa propre zone d'affichage 160x120.

Sommaire

L'objet *jit.op* vous permet d'effectuer des opérations mathématiques sur toutes les données d'une matrice Jitter en une seule fois. Vous pouvez effectuer des calculs sur les cellules de la matrice dans leur intégralité ou sur chaque plan séparément. L'opération mathématique que *jit.op* effectuera est déterminée par son attribut **op**, qui peut être saisi comme un argument d'attribut **@op [operator]** ou fourni par un message **op [operator]** dans l'entrée gauche. Pour les matrices à plans multiples (telles que les images couleur et les vidéos), vous pouvez spécifier l'opération pour chaque plan en fournissant une liste d'opérateurs (par exemple, **op pass * * ***) et vous pouvez fournir différentes valeurs scalaires pour chaque plan. Dans le tutoriel 9, vous verrez comment vous pouvez utiliser

une seconde matrice Jitter pour agir à la place d'un simple scalaire.

Vous pouvez définir la taille d'un objet *jit.pwindow* avec un message **size [width] [height]**. *jit.pwindow* fera de son mieux pour s'adapter à la taille de la matrice qu'elle reçoit. Il dupliquera les données si la matrice entrante est plus petite que ses dimensions, et ignorera certaines données si la matrice entrante est plus grande que ses propres dimensions. La plupart des objets Jitter font de leur mieux pour s'adapter aux dimensions, au type et au nombre de plans de la matrice qu'ils reçoivent. Dans le cas de *jit.op*, il n'a pas de dimension spécifique propres, il s'adapte donc aux caractéristiques de la matrice entrante.