49-Espaces colorés

> *Remarque*: certaines techniques décrites dans ce didacticiel sont obsolètes. Il est recommandé aux utilisateurs d'utiliser *jit.movie* avec **output_texture** activé au lieu de uyvy colormode pour l'efficacité de téléchargement des images de film vers le GPU. Consultez l'article *GL Texture Output* pour plus d'informations.

Afin de générer et d'afficher des matrices de données vidéo dans Jitter, nous faisons quelques hypothèses sur la façon dont l'image numérique est représentée. Pour de nombreuses utilisations typiques (couvertes par les didacticiels précédents), nous codons les images couleur dans des matrices à 4 plans de type **char**. Ces plans représentent les canaux de couleur alpha, rouge, vert et bleu de chaque cellule de cette matrice. Ce type de représentation des couleurs (ARGB) est utile car il correspond à la fois à la façon dont nous voyons les couleurs (par le biais des récepteurs de couleur dans nos yeux réglés sur le rouge, le vert et le bleu) et à la façon dont les écrans d'ordinateur, les projecteurs et les téléviseurs les affichent. Le *tutoriel 5: ARGB Color* examine le raisonnement qui sous-tend ce système et explique comment vous manipulez généralement ces données.

Il est important de noter, cependant, que le système ARGB n'est pas la seule façon de représenter les informations de couleur sous forme numérique. Ce didacticiel examine l'une des différentes façons de représenter les informations sur les images couleur dans les matrices Jitter, ainsi qu'une discussion sur plusieurs alternatives disponibles pour différentes utilisations. En cours de route, nous examinerons un moyen simple et efficace de texturer la vidéo sur un plan OpenGL afin de tirer parti d'un post-traitement de l'image vidéo accéléré par le matériel.

> Configuration logicielle requise: afin d'utiliser **uyvy colormode** dans Jitter sous QuickTime version 6.5 et antérieure, votre média doit être compressé avec un codec qui utilise un espace de couleur YUV (décrit en détail ci-dessous). Les supports QuickTime compressés avec des codecs utilisant un espace colorimétrique RVB (par exemple des fichiers PICT ou des fichiers vidéo RVB planaires) ne peuvent pas être décompressés par *jit.movie* dans **uyvy colormode** utilisé dans ce didacticiel. La version 7 de QuickTime et les suivantes vous permettront de travailler dans **uyvy colormode**, quel que soit l'espace colorimétrique dans lequel votre média est encodé.

• Ouvrez le patch du didacticiel.

À première vue, ce patch ressemble beaucoup à celui que nous avons utilisé dans le *didacticiel 12*: ***Color Lookup Tables***. Il lit un fichier dans un objet *jit.movie*, envoie les matrices dans un objet *jit.charmap*, où nous pouvons modifier le mappage des couleurs des différents plans de manière arbitraire en créant une matrice (**themap**) qui sert de table de conversion des couleurs. La matrice traitée est ensuite affichée.

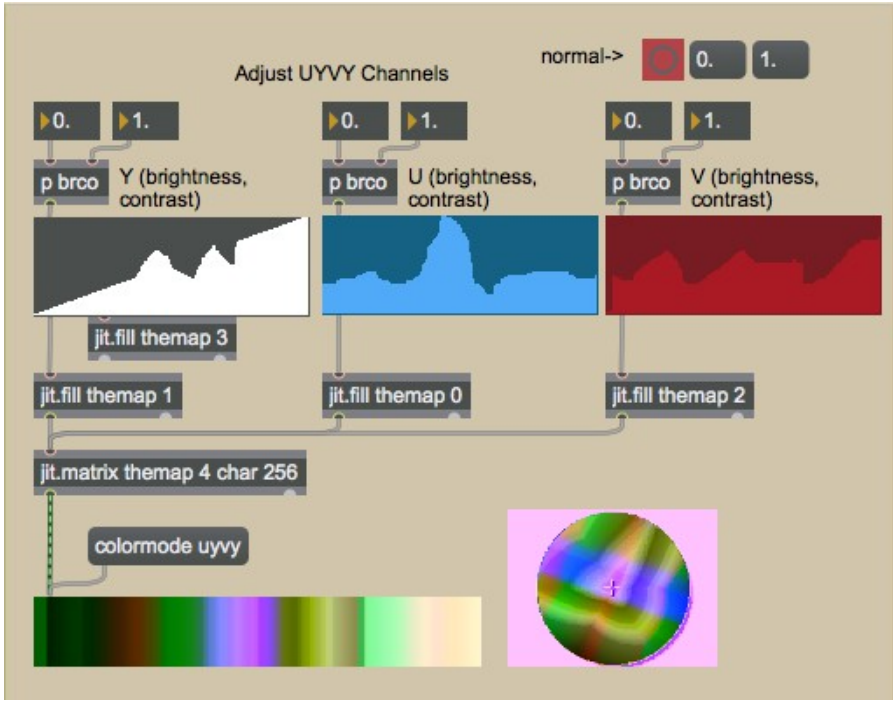
• Cliquez sur la boîte de *message* intitulée **read colorwheel.jpg**. Cliquez sur le *toggle* intitulé ***Display*** connecté à l'objet *qmetro* en haut du patch. Vous devriez voir la roue de couleur apparaître à la fois dans la fenêtre *jit.pwindow* et dans la fenêtre créée par l'objet *jit.window*.

Bien que ce ne soit pas immédiatement évident (encore), la matrice d'image de notre patch est générée et manipulée selon un système de couleur différent de la cartographie ARGB à laquelle nous sommes habitués. L'objet *jit.movie* de ce patch transmet les matrices en utilisant **uyvy colormode**. Cela signifie que la chaîne de traitement d'image travaille avec des données dans un espace colorimétrique différent de celui dans lequel nous travaillons habituellement, appelé YUV 4: 2: 2. En plus de transmettre la couleur selon un système de coordonnées différent de celui que nous

utilisons habituellement (YUV au lieu d'ARGB), ce mode de transmission utilise une technique appelée ***sous-échantillonnage chroma*** pour réduire (de moitié!) la quantité de données transmises pour une image de taille donnée.

Recherche de couleur avec une touche d'originalité

• Essayez de manipuler les objets *multislider* à droite du patch. Remarquez qu'ils ont un effet sur la façon dont la couleur est cartographiée dans l'image. Vous pouvez avoir l'impression que les trois objets *multislider* correspondent à l'affectation du vert, du rouge et du bleu dans la matrice, respectivement. Essayez de mettre à zéro le *multislider* le plus à gauche (noir sur blanc) (c'est-à-dire de régler toutes ses valeurs à **0**). Notez que l'image disparaît. Cliquez sur le *button* intitulé **normal** pour tout remettre en place. Essayez de mettre à zéro les deux autres *multisliders* à leur tour, puis de les régler sur des lignes horizontales droites au milieu de leur plage.



Manipuler la couleur.

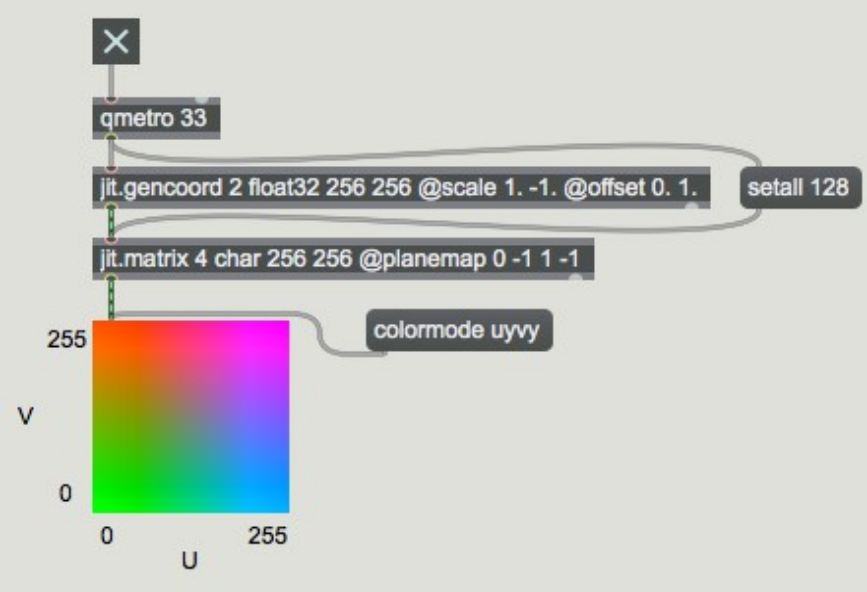
L'espace colorimétrique YUV est un système de couleurs luminance/chrominance - il sépare la luminosité d'une couleur donnée des informations chromatiques utilisées pour déterminer sa teinte. Il stocke la luminosité d'un pixel donné dans un canal de luminance (Y). Le canal U est ensuite créé en soustrayant le Y de la quantité de bleu dans l'image couleur. Le canal V est créé en soustrayant le Y de la quantité de rouge dans l'image couleur. Les canaux U et V (représentant la chrominance) sont ensuite mis à l'échelle par des facteurs différents. Par conséquent, des valeurs faibles de U et V exposeront des nuances de vert, tandis qu'une valeur moyenne constante des deux donnera une image en niveaux de gris. On peut convertir les valeurs de couleur de RVB en YUV en utilisant la formule suivante:

$Y = 0{,}299R + 0{,}587G + 0{,}114B$

$U = 0{,}492\ (B - Y)$

$V = 0{,}877\ (R - Y)$

Notez que les composants *U* et *V* de cet espace chromatique sont généralement signés (c'est-à-dire qu'ils peuvent être des nombres négatifs si la luminosité dépasse la quantité de bleu ou de rouge, comme c'est le cas avec des teintes telles que l'orange, le vert et le cyan). Les matrices Jitter stockent des données **char** non signées, de sorte que les valeurs U et V sont représentées dans la plage de 0 à 255, 128 étant le point central de l'espace chromatique.



Espace colorimétrique YUV avec une valeur Y de milieu de gamme constante.

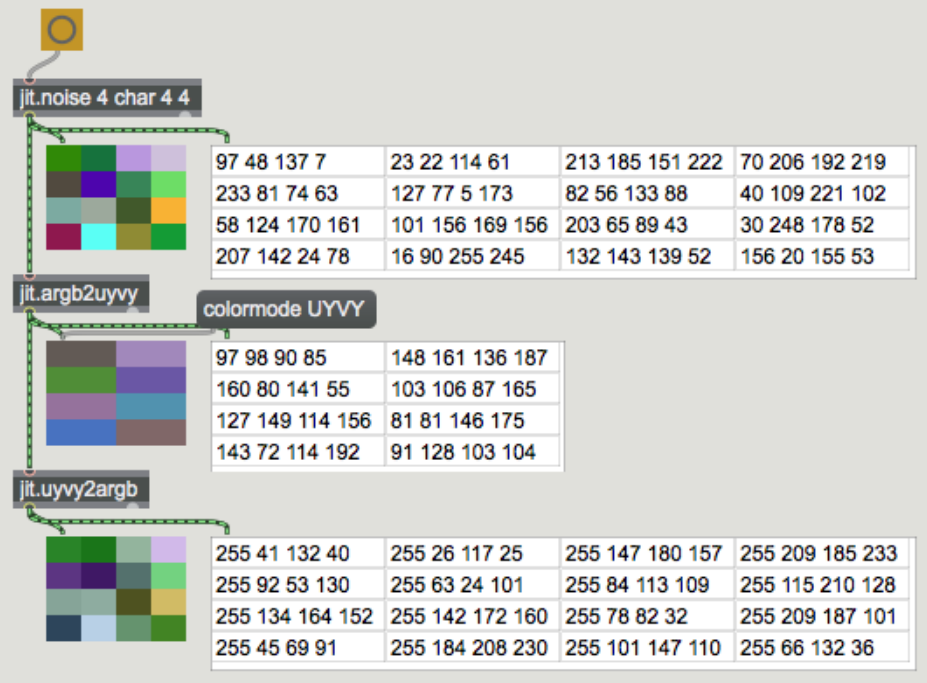
L'implémentation spécifique de l'espace de couleurs YUV utilisé dans notre patch de Tutoriel est appelé YUV 4: 2: 2. Les objets Jitter qui doivent interpréter des données matricielles comme de la vidéo (par exemple, *jit.movie*, *jit.pwindow*, etc.) peuvent générer et afficher cet espace de couleurs lorsque leur attribut **colormode** est défini sur **uyvy**. Ce **colormode** utilise ce qu'on appelle le ***sous-échantillonnage chromatique*** pour stocker deux pixels de couleur adjacents dans une seule cellule (appelée «macro-pixel»). Nos yeux étant plus sensibles aux fines gradations de luminosité qu'aux couleurs, il s'agit d'un moyen efficace d'effectuer une réduction des données sur une image, en divisant par deux la quantité d'informations nécessaires pour transmettre la couleur avec une précision raisonnable. Dans ce système, chaque cellule d'une matrice Jitter contient quatre plans qui représentent deux pixels horizontalement adjacents: le plan **0** contient la valeur U pour les deux pixels; le plan **1** contient la valeur Y pour le premier pixel; le plan **2** contient la valeur V pour les deux pixels; le plan **3** contient la valeur Y du second pixel. L'ordre des plans (uyvy) signifie que nous pouvons modifier la luminosité de l'image en ajustant les plans **1** et **3** (pour les colonnes de pixels alternées), mais que nous ne pouvons modifier la chrominance des pixels que par paires (en ajustant les plans **0** et **2**) .

> *Note historique*: Le codage couleur par luminance-chrominance (où la luminosité de l'image est transmise séparément de la teinte ou de la chrominance de l'image) trouve ses racines dans l'histoire de la télédiffusion couleur. Lorsque les téléviseurs couleur ont été introduits aux États-Unis en 1953, il était nécessaire de fournir un moyen aux consommateurs de télévision équipés de téléviseurs monochromes (noir et blanc) de pouvoir continuer à regarder les programmes. Il a donc été décidé d'ajouter simplement des informations de couleur (sous la forme d'une sous-porteuse) au signal de diffusion original, qui contenait déjà la luminosité de l'image en niveaux de gris. Le résultat a été appelé YIQ (pour luminosité-intermodulation-quadrature), et c'est l'espace couleur utilisé dans la télévision couleur NTSC. L'équivalent sur les systèmes de télévision PAL est l'espace colorimétrique YUV dont il est question ici.

L'illustration suivante montre comment la conversion d'ARGB en UYVY est gérée dans Jitter. Notre objet *jit.movie* effectue cette conversion pour nous lorsque cela est nécessaire (voir l'encadré ci-dessous), mais les objets *jit.argb2uyvy* et *jit.uyvy2argb* convertiront n'importe quelle matrice entre les espaces de couleurs. Notez que le canal alpha est perdu lors de la conversion et que les informations chromatiques sont moyennées sur des paires de cellules horizontales dans l'original ARGB, ce qui crée une légère perte d'informations chromatiques.

> Comme le système de couleurs UYVY utilise un macro-pixel, chaque cellule de la matrice représente en fait deux pixels horizontalement adjacents dans l'image; par exemple, une image de 320 x 240 pixels devient une matrice de 160 x 240 cellules lorsqu'elle est sortie en UYVY. Lors du traitement de ces matrices, il est important de garder cela à l'esprit, car les objets Jitter qui traitent les matrices basées sur des informations ***spatiales*** (par exemple, ceux qui font de la mise à l'échelle, de la rotation, de la convolution, etc.) traiteront ces paires de pixels comme une seule unité. Lorsque vous travaillez avec ces types de processus, il peut être plus facile de travailler dans un mode de couleurs qui utilise un pixel de pleine résolution, par ex. ARGB ou AYUV (un espace de couleur YUV de pleine résolution également supporté par Jitter).

(notez qu'un nouveau canal alpha vide est également créé).



Une grille 4x4 de valeurs aléatoires converties d'ARGB en UYVY et vice-versa.

> De nombreux codecs vidéo commerciaux utilisent YUV 4: 2: 2 (ou des systèmes de sous-échantillonnage chromatique similaires tels que 4: 1: 1 et 4: 2: 0) comme format vidéo natif. Par conséquent, *jit.movie* peut décompresser ces fichiers plus rapidement dans le mode de couleurs uyvy que lors de la sortie de matrices au format normal ARGB. Les matrices ainsi générées sont également deux fois plus petites, ce qui donne un gain de performances à tous les patchs Jitter qui peuvent tirer parti de ce système. Les codecs tels que Photo-JPEG, DVCPRO et NTSC DV utilisent tous une forme de codec YUV sous-échantillonné comme format de couleur natif.

Avec ceci en tête, nous pouvons comprendre le traitement qui se passe dans notre patch. L'objet *jit.movie* envoie des matrices au format **uyvy** 4 plans **char** à l'objet *jit.charmap*, qui traite les données et les envoie ensuite. Le *jit.fpsgui* nous dit que le **dim** des matrices qui lui sont envoyées

est 160x240, ce qui est logique maintenant que nous comprenons la réduction des données en macro-pixel qui accompagne le changement de **colormode**. De plus, nous pouvons maintenant voir pourquoi la matrice **themap** a un objet *jit.fill* alimentant les deux plans 1 et 3; ceux-ci correspondent aux deux valeurs Y dans le macro-pixel, que nous voulons partager avec la même table de conversion.

Teinte et saturation des couleurs

• Réglez les objets *multislider* sur leur courbe normale en cliquant sur l'objet *button* étiqueté ***normal***. Au-dessus des objets *patchers* étiquetés ***brco***, ajustez la boîte de *nombre* contrôlant le contraste du canal Y jusqu'à ce qu'elle indique **-1**.

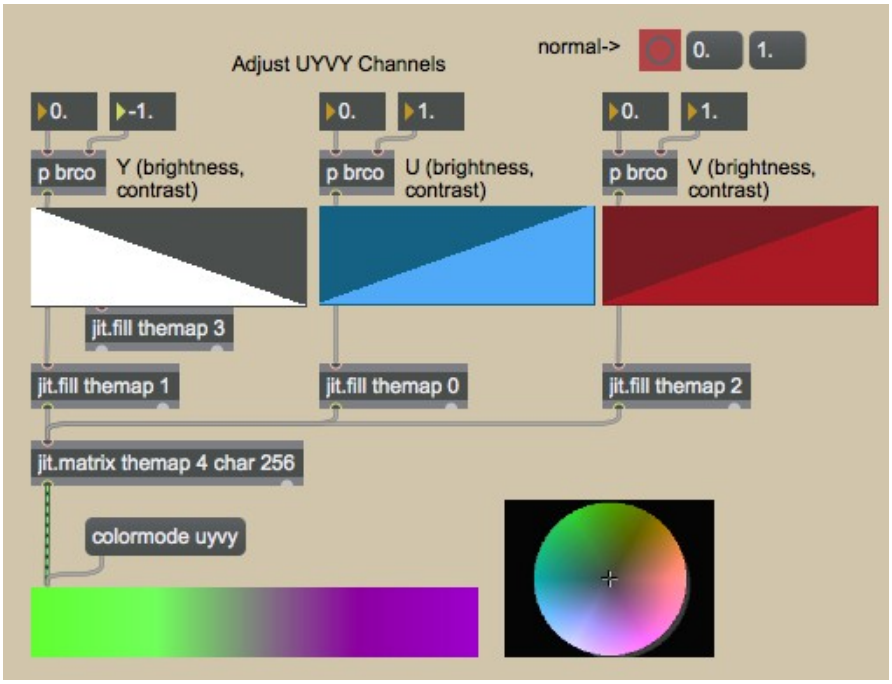
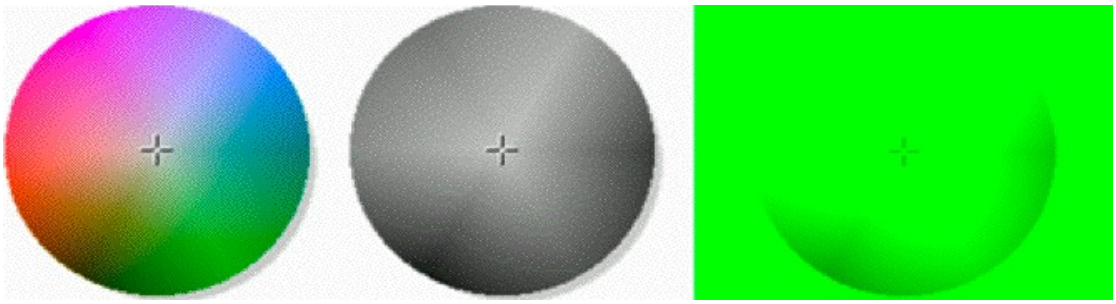


Table inversée de couleur.

Comme la luminosité est séparée de la chrominance dans notre nouvel espace colorimétrique, il est simple d'inverser la luminosité des pixels sans affecter leur teinte. Cette procédure serait plus complexe si l'on travaillait en l'ARGB.

• Normalisez à nouveau les objets *multislider* et ajustez les objets de boîte de *nombres* pour le contraste des canaux U et V. Essayez de les régler tous les deux sur **-1**, puis sur **0**. Normalisez leurs paramètres de contraste et réglez les valeurs de luminosité sur **-1**.



Différents paramètres U et V.

En inversant la recherche des couleurs des canaux U et V, nous effectuons une rotation de 180 degrés de la teinte de l'image. La définition de ces deux canaux à une valeur constante dé-sature l'image de manière à ce qu'elle apparaisse à un chroma ou une teinte constante, selon l'espace cartésien présenté plus tôt dans ce didacticiel. Une valeur de **0** dé-saturera l'image en niveaux de gris; une valeur de **-1** fera apparaître l'image entière en vert.

• Faites quelques dessins à main levée dans les objets *multislider*. Essayez de vous faire une idée de la façon dont différentes plages de la gamme de **char** entraînent des effets différents sur les différents canaux.

> Outre YUV 4: 2: 2 (**colormode uyvy**) et ARGB (**colormode argb**, la valeur par défaut pour la plupart des objets), il existe des objets Jitter qui génèrent des matrices dans un certain nombre d'autres espaces de couleur. Les exemples incluent **grgb** (un espace de couleurs RVB sous-échantillonné chromatiquement similaire à uyvy), **ayuv** (un espace de couleurs YUV pleine résolution avec un canal alpha), **luma** (un format de niveaux de gris à 1 plan) et **ahsl** (alpha, teinte, saturation, luminosité). Les objets de conversion sont généralement nommés jit.x2y, par exemple *jit.argb2uyvy*. De plus, l'objet *jit.colorspace* supporte la conversion vers et depuis une variété d'espaces de couleurs à 4 plans (y compris des approximations d'espaces de couleurs à virgule flottante nommés L * a * b *). Un bon point de départ pour obtenir plus d'informations sur les espaces de couleurs (et la représentation numérique de la couleur en général) est l'article Wikipédia sur le sujet.

Videoplane

• Remarquez que la sortie de *jit.pwindow* dans notre patch ne va pas directement vers un objet *jit.window*. Au lieu de cela, elle va vers un objet appelé *jit.gl.videoplane*. Cliquez sur la boîte de *message* intitulée **read track1.mov**. L'image traitée devrait se transformer en un film. Normalisez les tables de consultation des couleurs sur la droite en cliquant sur le *button* intitulé ***normal***.

L'objet *jit.gl.videoplane* texture la matrice Jitter envoyée dans son entrée sur un plan dans un contexte de dessin OpenGL. Notre contexte de dessin (**colorspace**) est piloté par l'objet *jit.gl.render* en haut du patch et est visible à travers la fenêtre de l'objet *jit.window*. Si vous devez revoir les bases du rendu OpenGL dans Jitter, jetez un œil au *Tutoriel 30*: ***Dessiner du texte 3D*** vous permettra de connaître les bases de la création d'un système de rendu. Une chose à noter est que l'attribut **ortho** de *jit.gl.render*, lorsqu'il est défini à **2**, rend notre scène dans une projection orthogonale (c'est-à-dire qu'il n'y a pas de sens de la profondeur). Notez également que l'objet *jit.gl.videoplane*, et non pas *jit.window*, doit recevoir l'instruction d'interpréter les matrices de texture comme **uyvy** par le biais de son attribut **colormode**. Dans notre patch, nous n'utilisons vraiment pas OpenGL pour la modélisation 3D; mais nous profitons de certaines caractéristiques du mappage de texture accéléré par le matériel.

• Cliquez sur la boîte de *message* contenant le texte **dim 16 16** (à côté de l'objet *jit.pwindow*).

Remarquez que l'objet jit.pwindow montre une image massivement sous-échantillonnée et pixellisée (elle est en fait traitée comme une matrice 8x16 car nous sommes toujours en mode **uyvy**). L'objet *jit.window,* par contre*,* montre une image où les pixels sont interpolés les uns dans les autres (l'effet est similaire au sur-échantillonnage dans un objet *jit.matrix* avec l'attribut **interp** défini sur **1**).

Interpolation matérielle d'une petite matrice appliquée comme une texture.

Cette interpolation se produit sur l'unité graphique de traitement graphique (GPU) du système et constitue l'un des nombreux avantages de l'utilisation d'OpenGL pour l'affichage vidéo, car elle n'entraîne aucune baisse de performances sur le CPU principal de notre ordinateur.

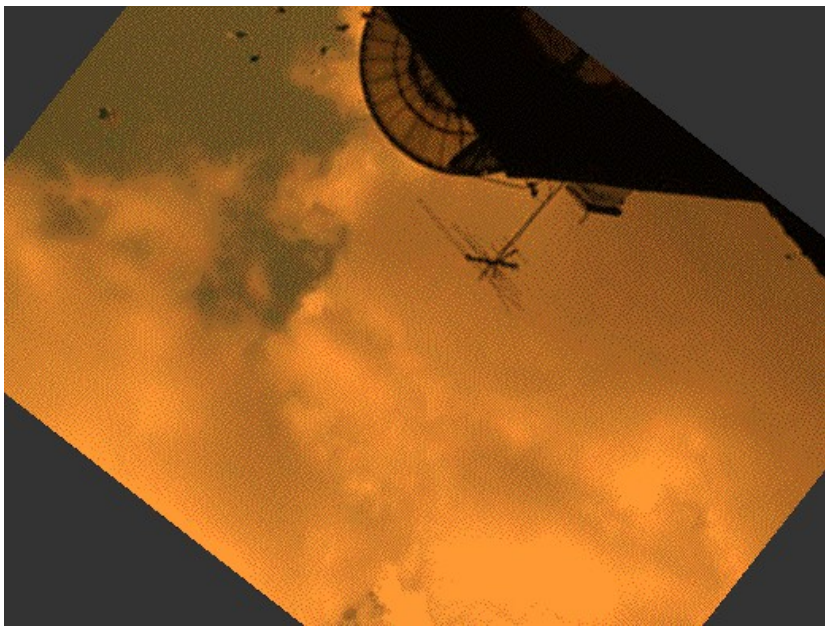
• Cliquez sur la boîte de *message* contenant le texte **dim 320 240**. Cela va ramener la taille des matrices à la résolution normale que nous avons utilisée: 320x240 pixels, ce qui donne 160x240 cellules par matrice en raison du mode **colormode**. À droite du patch, cliquez sur le *toggle* attaché à la boîte de *message* intitulée **fullscreen $1** (ou appuyez alternativement sur la touche ESC de votre clavier).

Lorsque vous envoyez un objet *jit.window* en mode plein écran, le *jit.gl.videoplane* sur-échantillonne la texture encore plus, vous donnant l'interpolation la plus lisse possible pour votre affichage.

• Appuyez sur la touche ESC pour déclencher à nouveau le *toggle*, et faire sortir *jit.window* de **fullscreen**.

Post-traitement de vidéoplane

• Ajustez les objets des boîtes de *nombres* rouge, vert et bleu attachés à l'objet *pak* connecté à l'objet *jit.gl.videoplane*. Notez que même si la texture appliquée est mappée dans **colorspace** YUV, le *jit.gl.videoplane* répond aux attributs **color** en virgule flottante RGBA. Manipulez la boîte de *nombre* intitulée ***rotation***.



Quelques exemples de traitement GPU sur videoplane.

Nous pouvons voir que les attributs **color** et **rotate** (ainsi que **scale**, **blend_enable**, etc.) de la plupart des objets OpenGL s'appliquent également à *jit.gl.videoplane*. Par conséquent, *jit.gl.videoplane* est un objet incroyablement utile pour le traitement vidéo, car il nous permet d'appliquer le traitement à l'image directement sur le GPU.

Résumé

L'objet *jit.movie* peut produire des matrices dans un certain nombre d'espaces de couleurs autres que ARGB. L'espace colorimétrique YUV 4: 2: 2 peut être utilisé en définissant l'attribut **colormode** des objets qui le supportent (*jit.movie*, *jit.pwindow* et *jit.window*) à **uyvy**. Le **colormode uyvy** a l'avantage d'utiliser un sous-échantillonnage chromatique de macro-pixels pour réduire de moitié le débit de données, ce qui permet de traiter les matrices plus rapidement dans la chaîne de traitement matriciel Jitter. Étant donné que la sortie de données dans cet espace colorimétrique est toujours constituée de 4 plans d'informations **char**, des objets standard tels que *jit.charmap* peuvent être utilisés pour manipuler la matrice, bien qu'avec des résultats différents.

L'objet *jit.gl.videoplane* accepte les matrices (y compris les matrices **uyvy**) comme des textures qui sont ensuite mappées sur un plan dans le contexte de dessin OpenGL nommé par l'objet. Le traitement accéléré par le GPU de l'image peut donc être effectué directement sur le plan, y compris la coloration, le mélange, la transformation spatiale, etc. Dans le *didacticiel 41*: ***Shaders***, nous avons vu comment appliquer des algorithmes de traitement entiers aux objets dans le contexte de dessin, élargissant encore les possibilités d'utilisation du GPU pour le traitement dans Jitter.