Hydra Functions

Readme!

Ce PDF est le fruit de deux « intelligences » la mienne, très modeste, et celle de l'Intelligence Artificielle. En effet, mon projet de compiler les fonctions disponibles dans Hydra pour lesquelles je cherchais souvent un complément d'explication (j'allais dire d'inspiration) tant la page dédiée sur le site d'Hydra hydra functions est à mon humble avis trop concise et quelque peu avare de commentaires (peut-être d'exemples aussi!).
J'ai donc sollicité l'aide précieuse de L'IA (ChatGPT) en l'occurence afin de satisfaire ma naturelle curiosité en ce que concerne ce programme extraordinaire qu'est hydra video synth et en particulier en ce qui concerne les fonctions dont nous nous servons pour faire vivre toutes ces créations visuelles!
Ce document est loin d'être parfait et se veut ouvert à toutes les améliorations que lui proposeront ses lecteurs: ne maitrisant pas parfaitement cet éditeur de texte, certains présentations ne sont pas idéalement optimisées ce dont vous vous rendrez compte par vous même, mais il a (comme on dit) le mérite d'exister!
Parfois, la mise en page dans Pages m'a posé problème sans que je puisse trouver une solution. J'avoue ne pas très bien maîtriser ces éditeurs de texte!
Certains exemples donnés par chatGPT ne sont pas justes et demandent à être corrigées (ce que j'ai fait bien-sûr) et améliorés. N'hésitez pas à devenir contributeur en améliorant ce document qui rendra, j'espère, à la collectivité! Merci de votre lecture et peut-être contribution! :-))
Beryann parker, live codeur et membre de Toplap Strasbourg/France
https://linktr.ee/beryann.parker

This PDF is the fruit of two "intelligences" - mine, very modestly, and that of Artificial Intelligence. Indeed, my project was to compile the functions available in Hydra, for which I was often looking for further explanation (I was going to say inspiration) as the dedicated page on the Hydra website hydra functions is in my humble opinion, too concise and somewhat lacking in commentary (and perhaps examples too!).
So I enlisted the invaluable help of AI (ChatGPT) to satisfy my natural curiosity about this extraordinary program, hydra video synth, and in particular about the functions we use to bring all these visual creations to life!
This document is far from perfect, and is open to any improvements suggested by its readers: as I don't master this text editor perfectly, some presentations are not ideally optimized, which you'll see for yourself, but it has (as they say) the merit of existing!
Sometimes the layout in Pages gave me problems but I couldn't find a solution. I admit that I don't master these text editors very well!
Some of the examples given by chatGPT aren't quite right and need to be corrected (which I've done, of course) and improved. Don't hesitate to become a contributor and improve this document, which I hope will give back to the community! Thank you for reading and perhaps contributing! :-))
Beryann parker, live coder and member of Toplap Strasbourg/France

https://linktr.ee/beryann.parker

Hydra video synth:
https://hydra.ojack.xyz/

The hydra book:
https://hydra-book.glitch.me/#/

5

SOURCES

NOISE

Dans Hydra, le mot-clé `noise` est une fonction générative puissante qui crée des motifs de bruit visuel souvent utilisés pour ajouter de la texture ou générer des formes abstraites dans des compositions visuelles.

Voici un aperçu des principaux aspects et paramètres de `noise` dans Hydra :

1. **Génération de motifs aléatoires** : La fonction `noise` génère des motifs visuels aléatoires basés sur des algorithmes de bruit (par exemple, le bruit de Perlin ou le bruit simple). Ce type de bruit crée des dégradés et des motifs organiques, souvent avec des formes douces et fluides, contrastant avec des motifs géométriques ou réguliers.

2. **Paramètres de `noise`** :

   - **scale** : ce paramètre contrôle la densité ou la résolution du bruit. Par exemple, une valeur de `scale` élevée produira un bruit détaillé et fin, tandis qu'une valeur basse génèrera des motifs de bruit plus larges et plus grossiers.

   - **offset** : contrôle le déplacement ou la position du motif de bruit, ce qui peut être utile pour animer des effets de bruit en modifiant progressivement l'offset au fil du temps.

3. **Utilisations typiques de `noise` dans Hydra** :

   - **Textures et fonds** : le bruit est souvent utilisé comme texture de fond pour ajouter une dimension organique à une scène.

   - **Modulation de couleurs** : `noise` peut être appliqué aux couleurs d'autres objets ou visuels pour leur donner une variation subtile et dynamique.

   - **Effets de mouvement** : en animant les paramètres de `noise`, il est possible de créer des effets de mouvement fluides et naturels, similaires à des vagues, des nuages ou même des éclats d’énergie.

Exemple d'utilisation de `noise` dans Hydra

Voici un exemple simple montrant comment utiliser `noise` pour générer un fond texturé animé :

```
noise(3)          // Le paramètre détermine l'échelle du bruit
  .color(0.5, 0.3, 0.9)  // Applique une couleur au motif
  .modulate(noise(1), 0.2) // Modulation pour plus de complexité
  .out()          // Envoie le résultat de l'animation à la sortie visuelle
```

Dans cet exemple, `noise(3)` définit un bruit avec une échelle moyenne. En appliquant `.color` et `.modulate`, on peut introduire des variations supplémentaires, et `.out()` affiche le résultat.

Avis

L'utilisation de `noise` dans Hydra peut s'avérer extrêmement efficace pour créer des animations abstraites et organiques. Les modulations successives permettent de simuler des textures de type biologique, ce qui en fait un outil précieux pour ceux qui explorent des visuels plus expressifs et fluides dans leurs créations.

VORONOI

Dans Hydra, la fonction `voronoi()` génère un effet visuel inspiré des diagrammes de Voronoï. Les diagrammes de Voronoï divisent l'espace en régions autour de points centraux, créant un effet de mosaïque où chaque région représente la zone d'influence d'un point particulier. Ce type de diagramme est couramment utilisé en mathématiques, en sciences naturelles et dans la visualisation de données pour modéliser des structures spatiales, mais Hydra en exploite le rendu pour produire des textures et motifs visuels intéressants.

Utilisation de `voronoi()` dans Hydra

Dans Hydra, le `voronoi()` crée des textures géométriques aux formes organiques, souvent utilisées pour des effets de fond ou comme couches dans des compositions visuelles plus complexes. Voici les principaux paramètres et leur effet :

1. **Nombre de cellules (premier paramètre)** : Le premier argument du `voronoi()` contrôle le nombre de cellules dans le diagramme. Plus le nombre est élevé, plus le diagramme est dense et complexe, tandis qu'un nombre faible produit de grandes cellules distinctes.

2. **Vitesse (deuxième paramètre)** : Ce paramètre détermine la vitesse à laquelle les cellules se déplacent ou se transforment dans le temps, créant un effet d'animation dynamique.

3. **Linéarité des bords (troisième paramètre)** : Le troisième paramètre contrôle le niveau de contraste des bords entre les cellules. Une valeur plus élevée rendra les bords plus nets et définis, donnant un effet plus "ciselé" au diagramme.

Exemple d'utilisation

```javascript
voronoi(10, 0.3, 0.5).out()
```

Dans cet exemple :

- `10` définit un diagramme avec une densité modérée de cellules.

- `0.3` donne une vitesse d'animation lente.

- `0.5` adoucit légèrement les bords des cellules, ajoutant un effet visuel subtil.

Applications créatives

L'effet `voronoi()` peut être combiné avec d'autres fonctions dans Hydra pour créer des visuels abstraits et texturés. Par exemple, il est courant de l'utiliser en superposition avec des effets comme `osc()`, `noise()`, ou `rotate()` pour générer des compositions visuelles avec plus de profondeur et de complexité.

Exemple

```
voronoi(25, 0.1, 1.5)
  .modulate(osc(10, 0.5, 1))
```

```
.out()
```

Ici, le diagramme de Voronoï est modulé par une oscillation, ce qui crée une interaction entre les cellules et ajoute un effet de distorsion intéressant.

Mon avis

`voronoi()` dans Hydra est un excellent outil pour expérimenter avec des visuels de type mosaïque. Son effet est particulièrement pertinent pour les créations abstraites et génératives, surtout lorsqu'il est combiné avec d'autres fonctions d'Hydra. Pour les artistes de visuels en temps réel, il offre une flexibilité pour créer des motifs dynamiques, adaptables à la musique live ou aux performances visuelles.

OSC

Dans Hydra, la fonction `osc()` génère une onde sinusoïdale oscillante sous forme de texture visuelle, souvent utilisée pour créer des motifs de bandes régulières, d'ondulations, et d'effets psychédéliques. Cette fonction est fondamentale pour les visuels génératifs car elle produit des motifs en mouvement fluide, parfaits pour des créations en temps réel ou synchronisées avec de la musique.

Fonctionnement et Paramètres de `osc()`

La fonction `osc()` prend généralement trois paramètres principaux qui influencent le motif de l'oscillation :

1. **Fréquence (premier paramètre)** : Contrôle le nombre de bandes d'oscillation affichées à l'écran. Une fréquence plus élevée augmente le nombre de bandes par unité d'espace, créant un effet plus dense, tandis qu'une fréquence plus faible produit de larges bandes.

2. **Vitesse (deuxième paramètre)** : Contrôle la vitesse du défilement des bandes. Cette vitesse crée un effet de mouvement des bandes, souvent dans une direction horizontale ou verticale, et permet de donner un dynamisme à la composition.

3. **Amplitude ou Intensité de Saturation (troisième paramètre)** : Ajuste l'intensité des couleurs, influençant la saturation et le contraste des bandes d'oscillation. Avec une valeur élevée, les bandes sont plus saturées et visibles, tandis qu'une valeur plus faible peut donner des oscillations plus douces, presque transparentes.

Exemple de Base

```
osc(10, 0.1, 0.9).out()
```

Dans cet exemple :

- `10` définit une fréquence de 10 bandes sur l'écran.

- `0.1` indique une vitesse de défilement lente, presque fixe.

- `0.9` donne une intensité de saturation forte, ce qui rend les couleurs vives.

Applications de `osc()` dans les Compositions Visuelles

La fonction `osc()` est polyvalente et peut être utilisée seule ou combinée avec d'autres fonctions pour obtenir divers effets visuels :

1. **Modulation** : En combinant `osc()` avec une autre fonction (comme `voronoi()` ou `noise()`), vous pouvez utiliser l'oscillation comme une couche de modulation qui altère les propriétés visuelles de la fonction de base, ajoutant un mouvement ondulant ou une texture dynamique.

Exemple :

```
osc(20, 0.3, 1.2)
  .modulate(noise(3))
  .out()
```

Ici, l'oscillation est modifiée par un bruit, donnant un effet de perturbation des bandes.

2. **Rotation et Couche de Profondeur** : La combinaison de `osc()` avec `rotate()` crée des effets de rotation spiralée sur les bandes, ajoutant une impression de profondeur et de mouvement en trois dimensions.

Exemple :

```
osc(15, 0.2, 1)
  .rotate(Math.PI / 4)
  .out()
```

Dans cet exemple, les bandes oscillantes sont inclinées, créant une dynamique visuelle intéressante.

3. **Effets Psychédéliques et Distorsions** : Pour des effets visuels plus intenses, on peut ajuster les paramètres de fréquence et de saturation et les combiner avec des fonctions comme `modulatePixelate()` ou `scrollX()`.

Exemple :

```
osc(50, 0.5, 2)
  .modulatePixelate(osc(10), 100)
  .scrollX(0.1)
  .out()
```

Cet exemple produit un effet psychédélique complexe, avec des bandes oscillantes pixelisées et un défilement horizontal.

Mon Avis

La fonction `osc()` est essentielle pour tout artiste travaillant avec Hydra, car elle permet de créer des visuels rythmés et fluides, idéaux pour des environnements immersifs ou des performances visuelles en temps réel. Son potentiel créatif est pratiquement infini, surtout lorsqu'on explore des combinaisons et variations avec d'autres fonctions. En manipulant les paramètres et en superposant des couches, il est possible de créer des visuels qui changent constamment, réagissent à la musique, et captivent l'attention du public.

Dans Hydra, la fonction `shape` génère une forme géométrique simple, souvent un polygone, qui peut être utilisée comme élément visuel de base dans une composition audiovisuelle. Elle est particulièrement flexible et permet de créer des effets visuels variés en ajustant ses paramètres.

SHAPE

Syntaxe et Paramètres

La fonction `shape` accepte plusieurs paramètres, chacun ayant une incidence sur la forme générée :

shape(sides, radius, smoothing)

1. **`sides`** : Ce paramètre détermine le nombre de côtés de la forme. Par exemple, une valeur de 3 crée un triangle, 4 crée un carré, et ainsi de suite. Pour des valeurs plus élevées, on obtient des polygones avec un plus grand nombre de côtés, voire des formes qui tendent vers le cercle pour des valeurs très élevées.

2. **`radius`** : Ce paramètre contrôle la taille relative de la forme. Une valeur de 0,5 correspond à une forme occupant environ la moitié de l'écran, tandis qu'une valeur de 1 permet à la forme de couvrir l'écran entier. Des valeurs supérieures à 1 peuvent entraîner un débordement de la forme en dehors de l'écran.

3. **`smoothing`** : Ce paramètre ajuste le lissage des bords de la forme. Une valeur plus élevée produit des bords plus doux, tandis qu'une valeur faible ou nulle produit des contours plus nets. Le

lissage peut être particulièrement utile lorsque vous cherchez à créer des formes plus esthétiques, surtout pour des cercles ou des polygones à plusieurs côtés.

Exemples d'Utilisation

1. **Forme de base (carré) :**

```
 shape(4, 0.3, 0.01)
.out()
```

   Cela génère un carré de taille moyenne avec un léger lissage.

2. **Cercle :**

```
   shape(100, 0.5, 1)
.out()
```

   En utilisant un nombre élevé de côtés (comme 100), on obtient visuellement un cercle.

3. **Effet kaléidoscope :**

```
   shape(6, 0.5, 0.1)
.repeat(4, 4)
   .out()
```

   En utilisant `repeat`, on crée un effet de répétition qui, combiné avec une forme hexagonale, peut ressembler à un kaléidoscope.

Applications et Combinaisons

La fonction `shape` est souvent combinée avec d'autres fonctions d'Hydra pour générer des visuels dynamiques et complexes. Par exemple :

- **`rotate`** pour faire tourner la forme,

- **`modulate`** pour créer des distorsions basées sur d'autres formes ou textures,

- **`scale`** pour ajuster la taille de manière dynamique.

Avis de ChatGPT

La fonction `shape` est un point de départ essentiel pour la création visuelle dans Hydra. Sa simplicité et sa flexibilité la rendent très accessible, même pour les débutants, tout en offrant une profondeur qui peut être exploitée par les utilisateurs avancés pour concevoir des compositions visuelles sophistiquées. Elle permet non seulement de générer des formes esthétiques de base, mais peut aussi servir d'élément structurant pour des effets plus élaborés lorsqu'elle est combinée à d'autres transformations d'Hydra.

GRADIENT

Dans Hydra, la fonction `gradient` génère un dégradé de couleurs, ce qui en fait un outil essentiel pour créer des effets visuels fluides et dynamiques. Elle permet de manipuler les teintes de manière continue, générant des transitions de couleurs qui peuvent être modifiées pour ajouter de la profondeur et des mouvements subtils aux compositions audiovisuelles.

Syntaxe et Paramètres

La fonction `gradient` dans Hydra s'utilise simplement avec un seul paramètre qui contrôle le nombre de cycles de couleurs dans le dégradé :

```
gradient(speed)
.out()
```

1. **`speed`** : Ce paramètre détermine la vitesse à laquelle le dégradé se déplace, créant un effet d'animation. Une valeur positive fait défiler le dégradé vers la droite (ou dans le sens horaire si la forme est circulaire), tandis qu'une valeur négative le fait défiler vers la gauche (ou dans le sens antihoraire). Une valeur de zéro crée un dégradé statique.

Fonctionnement et Utilisation du Dégradé

Le `gradient` génère une suite de teintes (généralement des couleurs dans l'espace HSL) qui se déplacent en continu selon la vitesse spécifiée. Ce type de dégradé est une bande continue de couleurs changeantes, permettant de créer une ambiance dynamique ou des effets hypnotiques.

Exemple d'Utilisation

Dégradé de base :

```
  gradient(0.5)
.out()
```

Ce code génère un dégradé avec une vitesse modérée, défilant lentement vers la droite.

2. **Dégradé statique :**

```
gradient(0)
.out()
```

Cette version génère un dégradé fixe, sans animation de déplacement.

3. **Dégradé rapide :**

```
gradient(3)
.out()
```

En augmentant la vitesse, on crée un effet visuel intense où les couleurs changent rapidement, donnant un effet de pulsation.

Combinaisons avec d'Autres Fonctions Hydra

La fonction `gradient` est souvent utilisée avec d'autres opérations visuelles dans Hydra pour obtenir des effets plus sophistiqués. Par exemple :

```
noise()
  .add(gradient(0.5))
.rotate(Math.PI / 2)
.colorama(0.1)
.out()
```

- **`rotate`** : L'effet de rotation appliqué à un dégradé permet de créer des tourbillons de couleur ou des effets de rotation psychédéliques. Par exemple :

```
rotate(Math.PI / 2)
```

Ce code fait tourner le dégradé à angle droit pour obtenir un effet de défilement vertical.

- **`colorama`** : Cette fonction accentue l'impact des changements de couleur, souvent utilisée avec `gradient` pour obtenir des effets de changement de teinte rapide.

Avis de ChatGPT

`gradient` dans Hydra est un outil fondamental pour ajouter de la profondeur visuelle et de la fluidité à une composition. Il agit comme une "texture de couleur" que l'on peut superposer ou combiner avec d'autres fonctions pour créer une richesse de visuels. Bien qu'il soit simple dans son utilisation de base, ses possibilités de combinaison le rendent très puissant, en particulier lorsqu'il est utilisé avec des fonctions de modulation et de distorsion. Utilisé efficacement, `gradient` apporte à la fois un dynamisme et une dimension hypnotique qui rendent les compositions visuelles de Hydra captivantes et immersives.

SRC

`src` est une **source d'image ou de texture**. C'est un point de départ pour manipuler ou combiner des visuels dans une composition. Les sources dans Hydra incluent généralement des motifs générés par l'ordinateur, des caméras, ou d'autres entrées vidéo. Voici plus de détails sur `src`

Fonctionnement de `src`

- `src` est utilisé pour désigner une **source de texture** provenant d'un flux ou d'une entrée spécifique.

- Par défaut, il fait référence à la **première source vidéo ou caméra** connectée au système (ou à l'écran noir au départ d'une nouvelle session?).

- On peut également diriger `src` vers d'autres sources, comme une **vidéo générée dans Hydra** ou des flux externes.

Exemple de base

Supposons que vous souhaitez utiliser une caméra ou un flux vidéo comme source dans Hydra :

```
s0.initCam()

src(s0)

.out(o0)
```

Si une caméra est configurée comme source principale, elle sera accessible directement comme entrée visuelle.

Utilisation avancée

Vous pouvez combiner `src` avec d'autres fonctions pour appliquer des effets visuels :

1. **Ajouter des filtres :**

```
s0.initCam()
src(s0)
.invert()
.modulate(osc(2),0.5)
.out(o0)
```

invert inverse les couleurs du buffer `o0`.

2. **Appliquer des masques ou des effets de mixage :**

```
src(o0)
  .blend(osc(10, 0.1, 0.5), 0.5)
  .out()
```

Ici, `src(o0)` est mélangé avec une oscillation (`osc`) pour un effet dynamique.

3. **Créer des rétroactions :**

```
src(o0)
  .scale(1.01)
  .rotate(0.1)
  .out(o0)
```

Cela crée une boucle de rétroaction avec un effet de zoom et de rotation.

Liens avec d'autres concepts Hydra

- **Buffers** : Les buffers (par exemple `o0`, `o1`, etc.) sont souvent combinés avec `src` pour manipuler les contenus générés en temps réel.

- **Caméras externes** : Si vous branchez une caméra USB ou utilisez un flux vidéo en ligne, `src` peut capturer et intégrer ce contenu dans vos visuels.

En résumé, `src` est une brique fondamentale dans Hydra qui permet de définir et manipuler des sources visuelles. Il est souvent utilisé pour travailler avec des flux d'entrée et créer des compositions complexes dans un environnement de live-coding visuel.

SOLID

Dans **Hydra** la fonction ou méthode **`solid()`** est utilisée pour créer une couleur unie ou un fond uniforme dans un visuel. Elle génère une texture qui remplit l'écran ou la zone assignée avec une couleur constante. Voici une explication plus détaillée :

Syntaxe :

```
solid(r, g, b, a)
```

Paramètres :

1. **`r` (rouge)** : Intensité de la composante rouge (entre 0 et 1).

2. **`g` (vert)** : Intensité de la composante verte (entre 0 et 1).

3. **`b` (bleu)** : Intensité de la composante bleue (entre 0 et 1).

4. **`a` (alpha)** : Opacité de la couleur (entre 0 et 1, optionnel). Par défaut, l'opacité est 1 (opaque).

Utilisation simple :

Créer un fond rouge opaque :

```
solid(1, 0, 0).out()
```

Résultat : l'écran sera entièrement rouge.

Utilisation avancée :

1. **Changer de couleur avec des animations :**

On peut utiliser des fonctions comme `Math.sin()` pour animer les valeurs et créer des transitions fluides.

```
Solid(()=>Math.sin(time), ()=>Math.cos(time), 0.5)
.out()
```

Dans cet exemple, la couleur change dynamiquement en fonction du temps (`time`).

2. **Combinaison avec d'autres fonctions :**

La texture générée par `solid()` peut être combinée avec d'autres fonctions (par exemple, `osc`, `shape`, etc.) pour créer des effets plus complexes :

```
solid(0.1, 0.1, 0.8)
  .mult(osc(10, 0.5, 1))
  .out()
```

Cela produit un motif où la couleur uniforme est modulée par une onde oscillante.

3. **Fusion avec des textures :**

On peut mélanger `solid()` avec d'autres couches ou appliquer des opérations comme `add()`, `sub()`, ou `blend()` :

```
 shape(4,0.2,0.5)
  .blend(solid(0, 0, 0.5, 0.5), 0.2)
  .out()
```

Ici, une couche bleue semi-transparente est mélangée avec la sortie actuelle.

Applications typiques :

- **Fond de base** pour des compositions visuelles.

- Ajout d'une couleur spécifique pour mélanger avec des textures animées.

- Génération de transitions douces dans un visuel.

GEOMETRY

ROTATE

Dans **Hydra**, la fonction `rotate` est utilisée pour faire pivoter une texture ou une image sur l'axe Z, en appliquant une rotation dans un espace bidimensionnel (2D). Cela permet de transformer visuellement la direction ou l'orientation d'une texture sur l'écran.

Syntaxe de base

```
rotate(angle, speed)
```

Paramètres

- **`angle`** : un nombre représentant l'angle de rotation, exprimé en radians. L'angle peut être dynamique (utilisant par exemple une fonction comme `time` ou une valeur mathématique) ou statique.

On peut paramétrer une rotation permanente (exemple donné dans « hydra functions » :

```
osc(50).rotate( () => time%360 ).out(o0)
```

- **`speed`** : définit la vitesse de rotation de façon dynamique comme dans cet exemple donné dans « hydra functions » :

```
osc(10,1,1)
  .rotate( () => time%360, () => Math.sin(time*0.1)*0.05 )
```

```
  .out(o0)
```

Fonctionnement

1. **Rotation horaire** : si l'angle est positif.

2. **Rotation antihoraire** : si l'angle est négatif.

Exemple d'utilisation

Exemple 1 : Rotation statique

```
osc(10, 0.1, 1)
  .rotate(Math.PI / 4) // Rotation de 45 degrés (PI/4 radians)
  .out()
```

Ici, une onde oscillante est générée et tournée de 45°.

Exemple 2 : Rotation dynamique avec le temps

```
osc(10, 0.1, 1)
  .rotate(time % (2 * Math.PI)) // Rotation continue sur 360°
  .out()
```

Dans cet exemple, la texture tourne en continu, car la valeur d'angle change avec le temps.

Exemple 3 : Rotation combinée avec d'autres transformations

```
osc(10, 0.1, 1)
  .rotate(Math.sin(time) * Math.PI) // Oscillation de l'angle entre -180° et 180°
  .modulateRotate(osc(5).rotate(0.3))
  .out()
```

Ce code combine la rotation d'une texture avec une modulation dynamique.

Cas pratiques

- **Créer des animations fluides** où les textures tournent en boucle.
- **Effets psychédéliques** en couplant `rotate` avec des fonctions comme `modulate` ou `kaleid`.
- **Manipuler des motifs complexes** pour créer des illusions optiques ou des effets géométriques.

SCALE

Dans **Hydra**, la fonction `scale()` est utilisée pour ajuster la taille des éléments visuels générés par le système. Cela peut affecter des formes, des textures ou des images, en augmentant ou réduisant leur échelle dans l'espace visuel.

Fonctionnement de `scale()`

1. **Signature** :

```
scale(x, y, z)
```

- `x` : Échelle sur l'axe horizontal (largeur).
- `y` : Échelle sur l'axe vertical (hauteur).
- `z` : Échelle sur la profondeur (uniquement utilisé dans des contextes 3D).(?)

Si vous ne spécifiez qu'un seul argument, il sera appliqué uniformément sur tous les axes.

2. **Exemple simple :**

```
shape(4) // crée un carré
  .scale(0.5) // réduit l'échelle à 50%
  .out()
```

Ici, le carré sera plus petit qu'à sa taille par défaut.

3. **Échelles indépendantes :**

```
shape(4)
  .scale(1, 0.5) // largeur normale, mais hauteur réduite à 50%
  .out()
```

4. **Échelle dynamique :**

Vous pouvez utiliser des fonctions ou des variables pour animer l'échelle :

```
shape(3)
  .scale(() => Math.sin(time) * 0.5 + 1) // variation dynamique
  .out()
```

Utilisation créative

- **Zoom avant/arrière** : Modifier l'échelle pour créer un effet de zoom progressif.

- **Effets psychédéliques** : Associer `scale()` à des fonctions comme `modulate()` ou `rotate()` pour des animations plus complexes.

- **Textures dynamiques** : Ajuster l'échelle de textures ou d'images pour des transformations fluides.

En résumé, `scale()` est un outil puissant pour manipuler la taille des éléments visuels, que ce soit de manière statique ou dynamique. Combiné à d'autres fonctions, il permet de créer des visuels interactifs et impressionnants.

PIXELATE

Dans **Hydra**, la fonction `pixelate()` est utilisée pour appliquer un effet de **pixellisation** à une texture ou une sortie visuelle. Elle réduit la résolution de l'image en regroupant les pixels par blocs, créant ainsi un effet de style rétro ou artistique.

Fonctionnement de `pixelate()`

1. **Signature** :

```
pixelate(x, y)
```

- `x` : Nombre de "pixels" (blocs) sur l'axe horizontal.
- `y` : Nombre de "pixels" (blocs) sur l'axe vertical.

Si vous spécifiez un seul argument (`x`), il sera appliqué uniformément à l'axe horizontal et vertical.

2. **Effet visuel** :

La fonction prend la texture ou l'image actuelle et la subdivise en blocs de tailles définies par les arguments. Plus les valeurs de `x` et `y` sont petites, plus les blocs sont grands, et donc plus l'effet de pixellisation est marqué.

Exemples simples

1. **Pixellisation uniforme** :

```
shape(4)
  .pixelate(10, 10) // 10 blocs horizontaux et verticaux (?)
  .out()
```

Cet exemple pixellisera un carré avec des blocs de taille uniforme (?)

2. **Pixellisation asymétrique** :

```
voronoi(5)
  .pixelate(20, 5) // plus de blocs horizontaux que verticaux
  .out()
```

Ici, l'image sera pixelisée avec des blocs larges horizontalement mais fins verticalement.

3. **Pixellisation dynamique** :

Vous pouvez animer les paramètres pour créer un effet de mouvement :

```
osc(10, 0.1, 1)
  .pixelate(() => Math.sin(time) * 20 + 30, 15)
  .out()
```

Cela crée un effet où les blocs horizontaux changent de taille dynamiquement au fil du temps.

-### Utilisation créative

- **Effet rétro** : Simuler l'apparence des anciens écrans ou des graphismes en basse résolution.

- **Esthétique glitch** : Coupler `pixelate()` avec des fonctions comme `modulate()` ou `kaleid()` pour des visuels complexes.

- **Masquage des détails** : Réduire la résolution pour simplifier ou abstraire une texture.

- **Transitions intéressantes** : Faire varier `x` et `y` pour passer progressivement d'une texture claire à une texture pixellisée.

Exemple avancé : combinaison avec d'autres effets

```
osc(20, 0.1, 0.8)
  .pixelate(10, 10)
  .modulate(noise(3), 0.5) // ajoute une distorsion par un bruit
  .out()
```

Ici, `pixelate()` agit en amont, puis l'effet est mélangé avec un bruit dynamique pour un rendu unique.

En résumé, `pixelate()` est une fonction versatile et très utile pour jouer sur la résolution et l'abstraction visuelle.

REPEAT

Dans **Hydra**, la fonction `repeat` est utilisée pour dupliquer une texture ou une forme sur une grille, créant un effet de répétition en mosaïque. Cela peut être utile pour générer des motifs ou des structures visuelles complexes à partir d'une seule texture.

Syntaxe de `repeat`

```
repeat(x, y)
```

- **x** : nombre de répétitions (ou fréquence) de la texture dans la direction horizontale.
- **y** : nombre de répétitions dans la direction verticale.

Fonctionnement

La fonction modifie la texture de manière périodique, la répliquant un certain nombre de fois selon les valeurs fournies pour `x` et `y`. Plus les valeurs de `x` et `y` sont élevées, plus la texture sera divisée en petites parties.

Exemple simple

```
osc(10, 0.1, 1)
  .repeat(3, 2) // Répète 3 fois horizontalement et 2 fois verticalement
  .out()
```

- Ici, un oscillateur est généré, puis répété 3 fois en largeur et 2 fois en hauteur, créant une grille de motifs.

Paramètres dynamiques

Les valeurs passées à `repeat` peuvent être animées ou générées dynamiquement pour créer des effets visuels plus vivants. Par exemple :

```
osc(20, 0.1, 1)
  .repeat(Math.sin(time) * 5 + 5, Math.cos(time) * 5 + 5)
  .out()
```

- Les répétitions varient au fil du temps, rendant l'effet dynamique.

Combinaison avec d'autres fonctions

`repeat` est souvent combiné avec des fonctions comme `modulate`, `scale` ou `rotate` pour créer des motifs complexes :

```
osc(5, 0.2, 0.8)
  .repeat(4, 4)
  .rotate(0.5)
  .modulate(osc(10), 0.3)
  .out()
```

- Cela produit une répétition avec rotation et modulation pour enrichir visuellement le résultat.

Points à noter

- Si `x` ou `y` est réglé à 1, il n'y aura pas de répétition dans cette direction.
- Des valeurs fractionnaires pour `x` ou `y` créent des distorsions intéressantes.

En résumé, `repeat` est un outil puissant dans Hydra pour introduire de la symétrie et des motifs répétitifs dans vos compositions visuelles.

REPEATX

Voici une nouvelle explication de la fonction `repeatX` dans Hydra, avec des exemples fonctionnels.

Fonction de `repeatX`

Les exemples sont à revoir !

`repeatX` sert à répéter une texture horizontalement sur la toile. Les paramètres sont les suivants :

```
.repeatX(frequency, offset)
```

- `frequency` : nombre de répétitions horizontales.
- `offset` *(facultatif)* : décalage horizontal entre les répétitions, exprimé en proportion de la largeur de la texture.

Nouveaux exemples

Exemple 1 : Répéter une onde sinusoïdale horizontalement

```
osc(10, 0.5, 1) // Génère une onde sinusoïdale
  .repeatX(4)   // Répète cette onde 4 fois sur l'axe X
  .out()        // Affiche le résultat
```

Ici, la texture de l'onde est répétée quatre fois horizontalement.

Exemple 2 : Ajouter un décalage aux répétitions

```
osc(10, 0.5, 1)  // Génère une onde sinusoïdale
  .repeatX(6, 0.2) // Répète 6 fois avec un décalage de 0.2
  .out()           // Affiche le résultat
```

Avec un décalage de `0.2`, chaque répétition est légèrement déplacée horizontalement, ce qui crée un effet de déphasage.

Exemple 3 : Animer le décalage avec le temps

Vous pouvez utiliser la variable `time` pour animer le décalage.

```
osc(5, 0.1, 1)              // Génère une onde sinusoïdale
  .repeatX(5, Math.sin(time) * 0.3) // Le décalage varie avec une sinusoïde
  .out()                     // Affiche le résultat
```

Le décalage est contrôlé par une fonction sinus qui varie avec `time`, ce qui donne un effet ondulatoire fluide.

Exemple 4 : Répéter une forme et changer la couleur

```
shape(4, 0.5, 0.1) // Génère une forme avec 4 côtés (un losange)
  .repeatX(3)       // Répète 3 fois sur l'axe X
  .color(0.2, 0.4, 0.8) // Applique une couleur
  .out()            // Affiche le résultat
```

Cet exemple montre une forme (losange) répétée trois fois horizontalement avec une couleur bleue.

Exemple 5 : Combiner `repeatX` avec `repeatY`

Pour obtenir une grille de répétitions, vous pouvez combiner `repeatX` avec `repeatY` :

```
shape(6, 0.3) // Génère une forme hexagonale
  .repeatX(5) // Répète horizontalement 5 fois
  .repeatY(4) // Répète verticalement 4 fois
  .out()      // Affiche le résultat
```

Cela produit une grille d'hexagones.

REPEATY

Dans **Hydra Video Synth**, la fonction `repeatY` est une transformation qui agit sur un flux visuel (ou texture). Elle permet de dupliquer l'image verticalement en répétant ses pixels sur l'axe Y, ce qui crée un effet de répétition verticale dans la composition visuelle.

Fonctionnement général de `repeatY`

La signature est typiquement la suivante :

```
repeatY(yScale, yOffset)
```

- `yScale` : Définit le nombre de répétitions verticales. Plus cette valeur est élevée, plus l'image sera divisée et répétée en fragments plus petits le long de l'axe Y. Une valeur de `1` signifie une seule répétition (pas de duplication), tandis qu'une valeur de `2` divise l'image en deux répétitions verticales.
- `yOffset` : Définit un décalage vertical appliqué après la répétition. Cela permet de faire glisser les répétitions verticales sur l'axe Y, créant un effet d'animation ou de décalage.

Exemple d'utilisation

```
osc(10, 0.1, 1)
  .repeatY(4, 0.2)
  .out()
```

- `osc(10, 0.1, 1)` : Génère un oscillateur visuel avec des bandes de fréquences.
- `repeatY(4, 0.2)` : Répète l'oscillateur 4 fois sur l'axe Y et applique un décalage de 0.2.
- `out()` : Envoie le résultat vers la sortie visuelle.

Effets visuels obtenus

- **Avec un yScale élevé** : Vous obtenez des répétitions de l'image plus fines, comme si elle était subdivisée en plusieurs segments verticaux.
- **Avec un yOffset animé** : Vous pouvez créer un mouvement fluide ou cyclique en modifiant dynamiquement cette valeur, par exemple :

```
osc(10, 0.1, 1)
.repeatX(6, Math.sin(time) * 0.5)  (préciser le rôle du Math.sin)
  .repeatY(6, Math.sin(time) * 0.5)
  .out()
```

Interaction avec d'autres fonctions

`repeatY` peut être combinée avec d'autres transformations pour enrichir les visuels :

- `scale` ou `rotate` : Pour ajuster l'échelle ou l'angle de l'image après répétition.
- `modulate` **ou** `blend` : Pour mélanger le flux répété avec d'autres flux.

En résumé

`repeatY` est un outil essentiel pour créer des motifs répétitifs verticaux, des effets de tuilage, ou encore pour manipuler les dimensions d'une composition visuelle dans **Hydra Video Synth**. Elle ouvre des possibilités créatives intéressantes, surtout lorsqu'elle est associée à des modulations dynamiques ou des animations.

KALEID

Fonctionnement général de `kaleid`

La signature de la fonction est la suivante :

```
kaleid(nSegments)
```

- `nSegments` : Le nombre de segments (ou divisions) utilisés pour le motif en kaléidoscope. Plus cette valeur est grande, plus l'effet sera complexe avec de nombreux motifs répétés. Une valeur basse (comme 2 ou 3) produira un effet avec peu de symétries, tandis qu'une valeur élevée (comme 10 ou 20) génèrera des motifs plus fins et détaillés.

Exemple d'utilisation

1. Application de base

```
osc(10, 0.1, 1)
  .kaleid(6)
  .out()
```

- `osc(10, 0.1, 1)` : Crée un oscillateur visuel avec des bandes colorées.
- `kaleid(6)` : Applique un effet de kaléidoscope avec 6 segments symétriques.
- `out()` : Envoie le résultat au flux de sortie.

2. Kaléidoscope dynamique

Pour un effet plus vivant, vous pouvez faire varier dynamiquement le nombre de segments à l'aide d'une fonction comme `Math.sin(time)` :

```
osc(20, 0.1, 2)
  .kaleid(Math.floor(6 + Math.sin(time) * 4))
  .out()
```

Dans cet exemple, le nombre de segments oscille entre 2 et 10, ce qui anime l'effet kaléidoscopique.

Effets visuels

- **Augmenter** `nSegments` : Rend l'image plus complexe, avec des motifs symétriques plus nombreux.
- **Diminuer** `nSegments` : Simplifie le motif, avec des formes plus larges et plus évidentes.
- **Combinaison avec d'autres transformations** :
  - `rotate` : Tourner l'image pour un mouvement fluide des motifs.
  - `modulate` **ou** `blend` : Ajouter des interactions avec d'autres flux visuels.
  - `repeat` **ou** `repeatX/Y` : Renforcer les motifs répétitifs pour un effet kaléidoscopique plus structuré.

Exemple avancé avec modulations

Vous pouvez combiner `kaleid` avec des oscillateurs, des modulations, et d'autres effets pour créer une composition complexe :

```
osc(40, 0.2, 1)
  .kaleid(8)
  .modulateRotate(osc(10).kaleid(4), 0.2)
  .out()
```

- `kaleid(8)` : Effet kaléidoscopique avec 8 segments.
- `modulateRotate` : Ajoute une rotation dynamique modulée par un autre oscillateur avec un kaléidoscope.

En résumé

La fonction `kaleid` dans **Hydra** est un puissant outil pour générer des motifs géométriques et des effets hypnotiques basés sur des symétries réfléchies. Elle est idéale pour créer des visuels abstraits,

des mandalas animés ou des compositions psychédéliques, surtout lorsqu'elle est combinée avec d'autres transformations ou des modulations dynamiques.

Le rôle de Math.floor()

Vous avez raison, la précision sur l'utilisation de `Math.floor` est importante pour bien comprendre son rôle dans l'exemple dynamique de la fonction `kaleid`. Voici une explication détaillée de son rôle et de son utilité dans ce contexte :

Rôle de `Math.floor` dans l'exemple

Dans cet exemple :

```
osc(20, 0.1, 2)
  .kaleid(Math.floor(6 + Math.sin(time) * 4))
  .out()
```

Fonctionnement :

1. `Math.sin(time)` : Génère une valeur oscillante entre -1 et 1 au fil du temps, créant un mouvement fluide et continu.
2. `6 + Math.sin(time) * 4` : Convertit cette oscillation en une plage de valeurs allant de **2 à 10**.
   - **Pourquoi ?** : La fonction `Math.sin(time)` est multipliée par 4, ce qui donne une amplitude de 8 (de -4 à +4). En ajoutant 6, le résultat est centré autour de 6, avec une oscillation entre 2 et 10.
3. `Math.floor` : Arrondit ce résultat vers le bas pour s'assurer que le paramètre transmis à `kaleid` soit un **entier**.
   - **Pourquoi un entier ?** : La fonction `kaleid(nSegments)` nécessite un nombre entier pour déterminer le nombre de segments. Sans `Math.floor`, la valeur oscillante pourrait être un nombre décimal, ce qui ne fonctionnerait pas correctement.

Sans `Math.floor`

Si on omet `Math.floor`, `kaleid` recevra une valeur décimale, comme **6.73**, ce qui peut produire des comportements inattendus ou des erreurs dans Hydra. Les segments ne seraient pas définis correctement, car `kaleid` utilise des divisions discrètes pour créer les symétries.

Exemple sans et avec `Math.floor`

Sans `Math.floor` :

```
osc(20, 0.1, 2)

  .kaleid(6 + Math.sin(time) * 4) // Peut causer des problèmes avec des valeurs non entières

  .out()
```

- Ici, `6 + Math.sin(time) * 4` produit des valeurs comme **7.56** ou **3.24**, ce qui peut créer des artefacts ou des effets inattendus.

Avec `Math.floor` :

```
osc(20, 0.1, 2)

  .kaleid(Math.floor(6 + Math.sin(time) * 4)) // Les valeurs sont arrondies à 7, 3, etc.

  .out()
```

- En utilisant `Math.floor`, on s'assure que `kaleid` reçoit uniquement des entiers (par exemple : 6, 7, 8), ce qui garantit un comportement fluide et prévisible.

En résumé

- `Math.floor` joue un rôle clé pour transformer une valeur flottante (comme 7.56) en un entier (comme 7).
- Cela est essentiel pour les fonctions comme `kaleid` qui nécessitent des **paramètres entiers** pour fonctionner correctement.
- L'ajout de `Math.floor` permet donc d'éviter des comportements imprévus et garantit que l'effet kaléidoscopique reste stable et cohérent.

SCROLL

Dans **Hydra**, la fonction `scroll()` est utilisée pour déplacer ou faire défiler un motif ou une texture sur l'axe horizontal (x) ou vertical (y). C'est une fonction très utile pour ajouter du mouvement aux visuels générés, créant un effet de translation.

Syntaxe :

```
.scroll(xAmount, yAmount, speedX, speedY)
```

Paramètres :

1. **xAmount** *(obligatoire)* :
    - Définit le décalage horizontal (axe x).
    - La valeur peut être positive (déplacement vers la droite) ou négative (vers la gauche).
2. **yAmount** *(obligatoire)* :
    - Définit le décalage vertical (axe y).
    - La valeur peut être positive (déplacement vers le haut) ou négative (vers le bas).
3. **speedX** *(optionnel)* :
    - Vitesse du défilement sur l'axe x.
    - Par défaut, aucune animation n'est appliquée si la vitesse n'est pas spécifiée.
4. **speedY** *(optionnel)* :
    - Vitesse du défilement sur l'axe y.
    - Par défaut, aucune animation n'est appliquée si la vitesse n'est pas spécifiée.

Exemple simple :

Voici un exemple où une texture est générée et défilée horizontalement et verticalement :

```
speed = 0.001

shape(4)

.scroll(0.1, 0.2,()=>Math.sin(time),()=>Math.sin(time)) /// Déplace lentement le motif sur les deux axes avec des vitesses de défilement sur les axes x et y

.out()
```

ou :

```
osc(20, 0.1, 1)

  .scroll(0.1, 0.2, 0.01, -0.02) // Défile horizontalement et verticalement à des vitesses différentes

  .out()
```

Utilisation avec des modulateurs :

Pour des effets plus dynamiques, on peut utiliser des oscillateurs ou d'autres fonctions pour moduler les paramètres de défilement :

```
shape(4)
```

```
  .scroll(() => Math.sin(time) * 0.1, () => Math.cos(time) * 0.1) // Animation
dynamique basée sur le temps

.diff(osc(10))

  .out()
```

Notes :

- La fonction `scroll()` ne "coupe" pas l'image ; elle crée un effet de répétition infinie grâce à la nature cyclique des motifs dans Hydra.
- Elle est souvent utilisée en combinaison avec d'autres fonctions (`rotate`, `modulate`, etc.) pour des visuels plus complexes.

Résumé : `scroll()` dans Hydra permet d'introduire du mouvement fluide et répétitif dans vos visuels en déplaçant des motifs ou des textures sur les axes horizontal et vertical, avec des options de contrôle de vitesse pour des animations continues ou dynamiques.

SCROLLX

Dans **Hydra**, la fonction `scrollX` est utilisée pour déplacer un motif, une texture ou un flux vidéo horizontalement (sur l'axe X) à travers la toile, créant ainsi un effet de défilement horizontal. Elle peut être combinée avec d'autres fonctions pour générer des animations complexes et dynamiques.

Syntaxe de base :

```
scrollX(amt, speed)
```

Paramètres :

1. **amt** *(float)* : la quantité de défilement horizontal, exprimée comme une proportion de la largeur de la toile. Par exemple :
   - `0.5` : l'image est décalée de la moitié de la largeur de la toile.
   - `1` : l'image est décalée de la largeur entière de la toile (répétée en boucle).
2. **speed** *(float)* : la vitesse du défilement, exprimée en cycles par seconde. Un nombre positif défile vers la droite, tandis qu'un nombre négatif défile vers la gauche.

Exemple simple :

```
osc(10, 0.1, 1)
  .scrollX(0.5, 0.1)
  .out()
```

- **`osc(10, 0.1, 1)`** génère une oscillation en forme d'onde.
- **`.scrollX(0.5, 0.1)`** déplace cette oscillation horizontalement de 50% de la largeur de la toile, avec une vitesse de défilement de 0.1 unités par seconde.

Exemple avec des modulations dynamiques :

```
osc(20, 0.05, 0.8)
  .scrollX(() => Math.sin(time) * 0.3, 0.2)
    .out()
```

Dans cet exemple :

- **`Math.sin(time) * 0.3`** fait osciller dynamiquement la position horizontale entre -0.3 et 0.3.
- Le défilement se produit également à une vitesse constante de 0.2 unités par seconde.

Utilisation avancée :

`scrollX` peut être combiné avec des textures, des effets comme `modulate`, ou encore des entrées vidéo (`s0` pour webcam, `s1` pour une source secondaire). Voici un exemple avancé :

```
voronoi(10, 0.5, 2)
  .scrollX(0.1, 0.05)
  .modulate(noise(3), 0.2)
  .out()
```

Ce code génère une texture en utilisant `voronoi`, applique un défilement horizontal lent avec `scrollX`, et modifie le résultat avec une texture bruitée.

En résumé :

- `scrollX` est parfait pour créer des effets de mouvement horizontal ou des transitions fluides.
- La combinaison avec des fonctions dynamiques ou des modulateurs ajoute de la profondeur à vos visuels.
- Vous pouvez l'utiliser pour synchroniser les mouvements avec de l'audio ou d'autres paramètres temporels.

SCROLLY

Same reasoning as for scrollX but on the y axis.

COLOR

POSTERIZE

Dans **Hydra**, la fonction `posterize` est utilisée pour réduire le nombre de nuances ou de niveaux dans une texture, créant ainsi un effet d'affiche (ou "posterized"), où l'image semble divisée en zones de couleurs ou de valeurs bien distinctes, sans transitions graduelles. Cela peut produire un rendu graphique, similaire à une image avec un nombre limité de couleurs.

Syntaxe de `posterize`

```
posterize(steps, gamma)
```
Paramètres :

1. **`steps`** *(nombre)* :
   Définit le nombre de niveaux (ou de "paliers") dans chaque canal de couleur. Une valeur basse (par exemple, 2 ou 3) produit un effet très segmenté avec peu de nuances, tandis qu'une valeur élevée conserve plus de détails.

2. **`gamma`** *(nombre) (optionnel)* :
   Ajuste le contraste des niveaux de couleur. Une valeur supérieure à 1 augmente le contraste entre les niveaux, tandis qu'une valeur inférieure à 1 les rend plus uniformes. Si ce paramètre est omis, un gamma par défaut est utilisé.

Fonctionnement :

L'effet `posterize` agit sur les canaux de couleur (rouge, vert et bleu) en quantifiant leurs valeurs en un certain nombre de niveaux. Chaque pixel est ajusté pour appartenir à l'un de ces niveaux prédéfinis.

Exemples d'utilisation :

Effet simple avec 3 niveaux :
```
osc(10, 0.1, 1)

 .posterize(3)
```

```
  .out()
```

Ici, l'oscillateur est réduit à seulement 3 niveaux de couleur dans chaque canal, créant un effet très graphique
Avec un gamma ajusté :

```
osc(20, 0.1, 1)
 .posterize(5, 2)
 .out()
```

Cela applique 5 niveaux de couleur avec un contraste accentué (gamma = 2), rendant les transitions entre les niveaux plus marquées.

`Combiné à d'autres effets :`

```
noise(5, 0.1
.posterize(4, 0.
 .kaleid(6)
.out()
```

Ici, l'effet d'affiche est combiné à un bruit et un effet de kaléidoscope, produisant un visuel complexe.

Modulation dynamique des niveaux : Vous pouvez animer dynamiquement le nombre de niveaux en utilisant une fonction comme `Math.sin` :

```
osc(10, 0.1, 1)
.posterize(() => Math.floor(Math.abs(Math.sin(time) * 10)), 1.5)
 .out()
```

`Cela fait varier le nombre de niveaux en fonction du temps, créant un effet en perpétuel changement.`

Résumé :

La fonction `posterize` est idéale pour créer des effets stylisés et graphiques en simplifiant les textures ou les images. Elle est particulièrement utile dans les performances visuelles pour obtenir un rendu unique et accrocheur, surtout lorsqu'elle est combinée avec d'autres transformations comme le défilement, le bruit ou les oscillateurs.

SHIFT

L'Intelligence artificielle me propose une explication qui me semble fausse! Je reprends les informations données dans la page dédiée aux fonctions d'Hydra déjà mentionnée plus haut.

A compléter!

shift(r = 0.5, g, b, a) // a = alpha/transparence

Shift décale (et « enveloppe ») les valeurs de r, g, b et/ou a !

// default

osc()

.shift(0.1,0.9,0.3)

.out()

Un exemple avec saturate():

osc(10, 0.1, 1)

shift(0.2,2,0.9,0.3)

.saturate(20)

.out()

INVERT

Dans **Hydra Video Synth**, la fonction `invert` est utilisée pour inverser les couleurs d'une image ou d'un flux vidéo. Plus précisément, elle soustrait chaque valeur de couleur de 1, ce qui correspond à une inversion dans l'espace de couleurs RGB.

Syntaxe

```
invert(amount)
```

Paramètre

- **amount** : Un nombre entre 0 et 1 qui contrôle l'intensité de l'inversion.

    - **1** : Inversion complète (chaque couleur est remplacée par son opposé).
    - **Valeurs intermédiaires** : Appliquent une inversion partielle, mélangeant les couleurs originales et inversées.

Fonctionnement

L'inversion des couleurs est réalisée en soustrayant chaque composante de couleur de 1. Par exemple, pour un pixel rouge pur (R: 1, G: 0, B: 0), l'inversion complète donne :

- R: 1 → 0 (inverse de 1),
- G: 0 → 1 (inverse de 0),
- B: 0 → 1 (inverse de 0), résultant en la couleur cyan (R: 0, G: 1, B: 1).

Exemple d'utilisation

```
osc(10, 0.1, 1)
  .invert(1)  // Inversion complète
  .out()
```

Dans cet exemple :

- Un oscillateur (`osc`) génère des formes d'ondes visuelles.
- La fonction `invert(1)` applique une inversion totale des couleurs.

Pour une inversion partielle :

```
osc(10, 0.1, 1)
  .invert(0.5)  // Inversion à 50%
  .out()
```

Combinaison avec d'autres fonctions

`invert` est souvent combinée avec d'autres fonctions comme `modulate`, `colorama` ou `kaleid` pour produire des effets visuels dynamiques et psychédéliques.

Par exemple :

```
osc(20, 0.05, 1)
  .colorama(0.5)
  .invert(0.7)
  .modulate(noise(3), 0.2)
  .out()
```

Ce code génère une composition complexe avec des couleurs inversées, des effets de modulation, et du bruit.

Applications créatives

- Créer des effets de contraste dramatique.
- Simuler un "mode négatif" pour des visuels.

- Introduire des variations visuelles dans des performances live.

En résumé, `invert` est une fonction puissante pour manipuler les couleurs et apporter un contraste dynamique ou des effets visuels inattendus dans Hydra.

CONTRAST

Dans **Hydra** le paramètre **contrast** permet de modifier le contraste des couleurs d'une texture ou d'une image générée dans Hydra. En termes simples, le contraste ajuste la différence entre les zones claires et sombres d'une image.

Fonctionnement

La méthode principale pour appliquer un contraste est la fonction `.contrast()`. Voici ses principales caractéristiques :

- **Syntaxe** :
  `.contrast(amount)`
- où `amount` est un nombre qui contrôle l'intensité du contraste.
- **Paramètres** :
  - `amount` : Une valeur numérique qui détermine le niveau de contraste.
    - Une valeur de **1** correspond au contraste d'origine (aucune modification).
    - Une valeur **supérieure à 1** augmente le contraste en renforçant les différences entre les couleurs claires et sombres.
    - Une valeur **inférieure à 1** diminue le contraste, rendant les couleurs plus uniformes.

Exemple simple

Voici un exemple de code dans Hydra pour illustrer l'utilisation de `.contrast()` :

```
osc(10, 0.1, 1)
  .contrast(1.5) // Augmente le contraste
  .out()
```

Dans cet exemple :

- `osc(10, 0.1, 1)` génère une onde avec des variations de luminosité.
- `.contrast(1.5)` augmente le contraste, rendant les zones lumineuses plus brillantes et les zones sombres plus sombres.

Effets combinés

L'effet de contraste peut être combiné avec d'autres transformations comme la saturation, le flou ou les effets de mélange pour créer des visuels dynamiques. Par exemple :

```
osc(20, 0.1, 0.8)
```

```
  .color(1, 0.5, 0.3)
  .contrast(2)
  .modulate(noise(3), 0.2)
  .out()
```

Applications créatives

- Accentuer les différences de textures pour des visuels plus percutants.
- Créer des styles graphiques spécifiques (par exemple, un effet high-contrast souvent utilisé dans l'art glitch).
- Préparer une image pour interagir avec des effets supplémentaires dans un workflow visuel complexe.

En résumé, `.contrast()` est un outil clé pour ajuster l'impact visuel des textures dans Hydra, permettant de jouer avec la dynamique entre les zones claires et sombres pour enrichir vos créations visuelles.

BRIGHTNESS

Dans **Hydra Video Synth**, la fonction **`brightness()`** est utilisée pour ajuster la luminosité d'une texture ou d'un flux vidéo dans votre composition. Cette transformation modifie la perception générale de l'intensité lumineuse de l'image sans affecter directement les couleurs ou le contraste. Voici une explication détaillée de son fonctionnement :

Fonctionnement :

- La fonction **`brightness()`** prend un seul argument numérique qui représente l'intensité de l'ajustement.
- La valeur par défaut est généralement **`0`**, ce qui signifie qu'il n'y a aucun changement appliqué à la luminosité.
- **Valeurs positives** augmentent la luminosité, rendant l'image plus claire.
- **Valeurs négatives** diminuent la luminosité, rendant l'image plus sombre.

Syntaxe :

```
brightness(amount)
```

Paramètre :

- **`amount`** : (Nombre) Contrôle la quantité d'ajustement de luminosité.
  - **0** : Aucun effet.
  - **Valeurs > 0** : Augmentent la luminosité.
  - **Valeurs < 0** : Réduisent la luminosité.

Exemple d’utilisation :

Augmenter la luminosité :

```
osc(10, 0.1, 1).brightness(0.5).out()
```

Dans cet exemple :

- L'oscillateur est utilisé comme source.
- La luminosité est augmentée de **0.5**, rendant l'image plus lumineuse.

Réduire la luminosité :

```
osc(10, 0.1, 1).brightness(-0.5).out()
```

Ici, la luminosité est réduite de **-0.5**, rendant l'image plus sombre.

Utilisation créative :

- **Créer des ambiances** : Ajuster la luminosité permet de donner une sensation d'intensité ou d'obscurité à une scène.
- **Moduler dynamiquement** : Associez **`brightness()`** avec des fonctions comme **`modulate()`** pour des variations dynamiques de lumière.

```
osc(10, 0.1, 1)
  .brightness(() => Math.sin(time) * 0.5) // Luminosité
oscillante
  .out()
```

Notes importantes :

- **Changements subtils** : Évitez des valeurs trop extrêmes pour conserver les détails de l'image.
- **Interaction avec d'autres effets** : La fonction peut être combinée avec d'autres transformations (e.g., **`contrast()`**, **`saturation()`**) pour des compositions plus complexes.

Si vous explorez d'autres manipulations visuelles, n’hésitez pas à expérimenter avec **`brightness()`** en combinaison avec des oscillateurs, des entrées vidéo, ou des modulations.

LUMA

Dans **Hydra Video Synth**, la fonction **`luma()`** sert à isoler ou manipuler une partie spécifique d'une texture ou d'un flux vidéo basée sur ses niveaux de luminosité. Elle agit comme un filtre permettant de sélectionner les zones de l'image en fonction de leur intensité lumineuse.

Fonctionnement de `luma()`

La fonction travaille sur le **canal de luminance** d'une texture, qui correspond à la luminosité ou à la clarté perçue des pixels. Les zones avec une luminosité qui correspond au seuil défini par l'utilisateur seront conservées, tandis que les autres seront atténuées (ou rendues transparentes).

Syntaxe :

```
luma(threshold, tolerance)
```

Paramètres :

1. **`threshold`** *(Nombre)* :
    - Définit le niveau de luminosité à conserver.
    - **0** : Conserve les zones très sombres.
    - **1** : Conserve les zones très lumineuses.
2. **`tolerance`** *(Nombre)* :
    - Contrôle la transition autour du seuil. Une valeur élevée adoucit la sélection des zones de luminance.
    - **0** : Aucun adoucissement (les zones en dehors du seuil sont entièrement transparentes).
    - **Valeurs plus élevées** : Crée une transition plus douce entre les zones visibles et transparentes.

Exemple d'utilisation :

Isoler les zones lumineuses :

```
osc(10, 0.1, 1).luma(0.7, 0.05).out()
```

Dans cet exemple :

- Les zones de l'image avec une luminance supérieure à **0.7** sont conservées.
- Une **tolérance** de **0.05** adoucit légèrement la transition.

Créer un masque avec les zones sombres :

```
osc(5, 0.1, 1).luma(0.2, 0.1).out()
```

Ici :

- Les zones sombres (avec une luminance inférieure à **0.2**) sont isolées.

- La tolérance de **0.1** adoucit les bords du masque.

Utilisation avancée :

1. **Avec une source vidéo ou webcam** :

```
s0.initCam()

src(s0).luma(0.5, 0.1).out()
```

```
- Cela conserve les zones d'une vidéo en direct avec une
luminosité proche de **0.5**.

2. **En combinaison avec des modulateurs** :

osc(10, 0.2, 1)
.luma(() => Math.sin(time) * 0.5 + 0.5, 0.1)
.out()
```

- Le **seuil** de luminosité varie dynamiquement en fonction du temps, créant un effet oscillant.

3. **Créer des masques dynamiques** : Vous pouvez utiliser **`luma()`** pour superposer ou masquer des textures en fonction de leur luminosité. Par exemple :

```
osc(20, 0.1, 1) .layer(noise(2))

 .luma(0.5, 0.05)

 .out()

---

### **Applications créatives :**
- **Créer des masques** : Isoler des parties spécifiques
d'une image pour y appliquer des effets ou les combiner avec
d'autres couches.
- **Effets dynamiques** : En variant le **threshold** ou la
**tolerance** de manière interactive, vous pouvez obtenir des
animations lumineuses captivantes.
- **Superposition d'éléments** : Utiliser `luma()` comme un
outil pour composer différentes sources ou flux dans Hydra.

---

### **Notes importantes :**
```

- L'effet fonctionne mieux avec des textures ayant une plage dynamique large (comme des oscillateurs ou des entrées vidéo bien contrastées).
- **Expérimentation** : Associez **`luma()`** avec d'autres fonctions comme **`contrast()`** ou **`brightness()`** pour ajuster les textures avant ou après l'application du filtre.

En résumé, **`luma()`** est un outil puissant pour travailler avec la luminosité et créer des visuels expressifs et précis.

TRESH

Dans Hydra, le logiciel de synthèse vidéo en temps réel conçu par Olivia Jack, la fonction `tresh` (abréviation de "threshold", ou seuil en anglais) est utilisée pour appliquer un effet de seuil sur une image ou une texture vidéo. **Cet effet transforme les valeurs des pixels selon un seuil défini, créant des zones binaires (?) basées sur leur luminosité ou d'autres paramètres**. Cela peut donner des effets visuels de type "posterisation" ou des contrastes très marqués.

Fonctionnement de `tresh`

La fonction `tresh` compare la luminosité (ou une autre propriété) des pixels à un certain seuil. Voici un exemple de sa syntaxe :

```
tresh(threshold, tolerance)
```

- **`threshold`** : Détermine la valeur de seuil. Les pixels ayant une valeur supérieure ou égale à ce seuil deviennent blancs (ou une autre couleur définie par les modifications du signal), tandis que les autres deviennent noirs.
- **`tolerance`** : Introduit une plage autour du seuil, permettant une transition plus douce entre les zones claires et sombres.

Exemples d'utilisation

Voici un exemple dans Hydra où on applique l'effet `tresh` :

```
osc(10, 0.1, 1)
  .tresh(0.5, 0.1)
  .out()
```

1. **`osc(10, 0.1, 1)`** : Génère un oscillateur (ondes visuelles) avec une fréquence de 10.
2. **`.tresh(0.5, 0.1)`** : Applique un seuil à 0.5 avec une tolérance de 0.1.
3. **`.out()`** : Envoie le signal au rendu visuel.

Autre exemple:

```
osc(30)
.layer(osc(15)
.rotate(1)
.thresh())
.out(o0)
```

Applications pratiques

- **Esthétique glitch ou minimaliste** : En simplifiant les textures avec des zones très contrastées.
- **Effets interactifs** : Quand il est combiné avec des sources audio ou d'autres entrées interactives.
- **Transformations stylisées** : Avec d'autres fonctions comme `modulate` ou `mult`, pour créer des visuels complexes.

Si vous travaillez avec Hydra dans un contexte créatif, `tresh` est une fonction puissante pour manipuler la clarté et le contraste de vos visuels. N'hésitez pas à la combiner avec d'autres opérations pour enrichir vos compositions !

COLOR

Dans **Hydra Video Synth**, la fonction ou méthode `color()` est utilisée pour générer une texture uniforme avec une couleur définie par ses trois composantes de base : **rouge (R)**, **vert (G)**, et **bleu (B)**. Elle est essentielle pour créer des bases visuelles ou ajouter des effets colorés sur d'autres textures. Voici les détails de son fonctionnement :

Syntaxe de `color()`

```
color(r, g, b, a)
```

Paramètres :

- **r (rouge)** : Une valeur entre `0` et `1` représentant l'intensité du rouge.
- **g (vert)** : Une valeur entre `0` et `1` représentant l'intensité du vert.
- **b (bleu)** : Une valeur entre `0` et `1` représentant l'intensité du bleu.
- **a (alpha)** *(optionnel)* : Une valeur entre `0` et `1` représentant la transparence (par défaut, la valeur est `1`, c'est-à-dire totalement opaque).

Fonctionnement :

- Lorsque vous utilisez `color()`, Hydra génère une texture remplie d'une couleur uniforme selon les valeurs spécifiées.
- Cette texture peut être utilisée directement ou combinée avec d'autres textures via des fonctions comme `blend()`, `add()`, ou `modulate()`.

Exemple d'utilisation :

1. **Couleur de base** :
   ```
   color(1, 0, 0).out()
   ```
2. Cela produit une couleur rouge pleine.
3. **Couleur avec transparence** :
4. .gradient().
5. `color(0, 1, 0, 0.5)`
6. `.out()`
7. **Cela crée une texture verte semi-transparente.**
8. **Effet combiné avec d'autres textures** :
9. `osc(10, 0.1, 1)`
10. `.color(1, 0, 0)`
11. `.out()`
12. Ici, l'oscillateur est coloré en rouge.
13. **Ajouter un gradient coloré avec `colorama()`** :
14. `osc(20, 0.1, 1)`
15. `.color(0.5, 0.5, 1)`
16. `.colorama(0.3)`
17. `.out()`
18.

Notes importantes :

- Les valeurs des couleurs suivent le modèle **RGB** (Rouge, Vert, Bleu) en échelle flottante entre `0` et `1`.
- `color()` est souvent utilisée comme un outil simple pour expérimenter avec les couleurs ou créer des effets visuels complexes lorsqu'elle est combinée à d'autres fonctions de Hydra.

SATURATE

Dans **Hydra Video Synth**, la fonction **`saturate()`** est utilisée pour ajuster la **saturation** d'une texture, c'est-à-dire l'intensité des couleurs présentes dans l'image ou la texture générée. Elle permet d'amplifier ou de réduire la vivacité des couleurs, ce qui est utile pour créer des effets visuels dynamiques et expressifs.

Syntaxe de `saturate()`

```
texture.saturate(amount)
```

Paramètres :

- **`amount`** : Un nombre (positif ou négatif) représentant l'intensité de la saturation appliquée.
  - **Valeurs positives** : Augmentent la saturation, rendant les couleurs plus vives.
  - **Valeurs négatives** : Réduisent la saturation, jusqu'à atteindre un effet désaturé ou totalement en niveaux de gris.
  - **Valeur par défaut** : Si aucune valeur n'est spécifiée, elle est considérée comme 0 (aucune modification).

Fonctionnement :

1. **Augmenter la saturation** :
   Lorsque vous appliquez un nombre positif, les couleurs deviennent plus intenses, créant un effet de sursaturation où les couleurs peuvent paraître "exagérées".

2. **Réduire la saturation** :
   Avec des nombres négatifs, les couleurs perdent leur intensité, donnant un effet terne ou monochromatique.

3. **Effet de contraste des couleurs** :
   Les textures combinées à `saturate()` peuvent produire des effets visuels captivants, surtout si elles contiennent une large gamme de teintes.

Exemples d'utilisation :

Augmenter la saturation d'une texture simple :

```
osc(10, 0.1, 1)
  .saturate(2)
  .out()
```

Cela rend les couleurs de l'oscillateur beaucoup plus vives.
Réduire la saturation :

```
osc(15, 0.2, 1)

  .saturate(-1)
  .out()
```

Cela désature la texture, rendant l'image presque en niveaux de gris.

Saturation combinée avec d'autres effets :

```
osc(20, 0.05, 0.8)

  .kaleid(5)
  .saturate(3)
  .modulate(osc(10, 0.1).saturate(-2))
  .out()
```

Ici, la saturation amplifie les contrastes visuels dans une texture modifiée par un oscillateur secondaire désaturé.
Transition vers un effet désaturé :

```
osc(25, 0.05, 1)

  .saturate(() => Math.sin(time) * 2) // Variation dynamique
avec le temps
  .out()
```

La saturation évolue dynamiquement avec une fonction sinus.

Applications pratiques :

- **Pour styliser les visuels** : Créez des effets éclatants ou adoucissez les couleurs selon l'ambiance recherchée.
- **Pour jouer avec les niveaux de détail** : Une saturation élevée peut rendre les textures plus agressives visuellement, tandis qu'une saturation faible peut générer un effet minimaliste.
- **Pour des transitions dynamiques** : Combinez `saturate()` avec des modulations temporelles pour des effets évolutifs.

En résumé, `saturate()` est un outil puissant pour jouer avec la richesse des couleurs dans Hydra et donne aux visuels une profondeur esthétique supplémentaire.

HUE

Dans **Hydra**, une plateforme de programmation pour le live coding visuel, la fonction **`hue`** est utilisée pour manipuler la teinte d'une image ou d'une texture. Elle permet de décaler les couleurs d'une composition visuelle en ajustant leur teinte sur le cercle chromatique, tout en préservant leur saturation et leur luminosité. Voici un aperçu de son rôle et de son fonctionnement :

Fonctionnement de `hue`

La méthode **`.hue()`** modifie la teinte de chaque pixel dans une image. La teinte est ajustée en fonction d'une valeur spécifiée, exprimée généralement sous forme d'un flottant entre -1 et 1.

- **Valeur positive** : Déplace les couleurs dans le sens des aiguilles d'une montre sur le cercle chromatique.
- **Valeur négative** : Déplace les couleurs dans le sens inverse des aiguilles d'une montre.
- **Valeur nulle (0)** : Ne change pas la teinte, laissant les couleurs inchangées.

Syntaxe

`hue(value)`

- **`value`** : Un flottant qui représente la quantité de décalage de teinte.
  Par exemple :
  - `hue(0.2)` : Décale les couleurs légèrement vers le rouge.
  - `hue(-0.3)` : Décale les couleurs légèrement vers le bleu/vert.

Exemple simple

Prenons une texture de base qui produit des vagues et appliquons un décalage de teinte :

```
osc(10, 0.1, 1)  // Génère une texture oscillante
  .hue(0.5)      // Décale la teinte de 50%
  .out()         // Envoie la sortie à l'écran
```

Effet dynamique

Pour créer un effet dynamique, vous pouvez animer la teinte en fonction du temps avec `time` :

```
osc(10, 0.1, 1)
  .hue(Math.sin(time) * 0.5)  // Animation continue sur la
teinte
  .out()
```

Combinaisons créatives

`hue` peut être combinée avec d'autres fonctions comme `modulate`, `invert` ou `brightness` pour enrichir les effets visuels.

```
osc(20, 0.1, 1)
  .hue(() => Math.sin(time) * 0.3)  // Variation dynamique
  .modulate(noise(3))              // Ajout d'une modulation
  .out()
```

Application pratique

- **Transitions douces** dans des performances live.
- **Création de palettes colorées vivantes** en boucle ou en animation.
- **Harmonisation des visuels** avec une musique ou un autre média en synchronisant le décalage de teinte avec des données externes.

En résumé, **`hue`** est un outil puissant pour la manipulation des couleurs dans Hydra, ajoutant une dimension dynamique et esthétique à vos compositions visuelles.

COLORAMA

Dans Hydra, `colorama` est une fonction conçue pour manipuler les couleurs d'un flux visuel en leur appliquant une rotation de teintes. Elle permet de donner un effet psychédélique ou kaléidoscopique à la composition, ce qui peut dynamiser les visuels en introduisant des changements subtils ou intenses dans les couleurs.

Description détaillée de `colorama`

La fonction `colorama` modifie essentiellement la teinte de chaque pixel d'une image en appliquant une rotation des couleurs. Elle agit comme un filtre qui balaie le spectre de couleurs (rouge, orange, jaune, vert, bleu, violet) de manière cyclique, créant des variations de teintes. En ajustant le paramètre de `colorama`, on contrôle la vitesse et l'intensité de cette rotation de teinte, influençant ainsi le rendu visuel final.

Paramètre principal de `colorama`

La fonction prend un seul paramètre principal qui contrôle l'intensité et la vitesse de rotation des couleurs :

- **paramètre `amount`** : il s'agit d'un nombre (positif ou négatif) qui représente le taux de changement de couleur. **Des valeurs plus élevées rendent la rotation des teintes plus rapide, tandis que des valeurs plus faibles produisent des changements plus subtils et lents. Une valeur négative inverse la direction de la rotation des couleurs.**

Exemple d'utilisation de `colorama`

Voici un exemple simple montrant comment appliquer `colorama` pour animer les couleurs d'un motif de bruit dans Hydra :

```
noise(3, 0.1)        // Crée un motif de bruit avec une certaine échelle et vitesse
  .colorama(0.5)      // Applique une rotation de teinte avec une intensité moyenne
  .out()              // Envoie le résultat vers la sortie visuelle
```

Dans cet exemple :

- `noise(3, 0.1)` génère un motif de bruit de texture.

- `.colorama(0.5)` ajoute un effet de décalage de teinte d'intensité moyenne. Les couleurs changent de manière cyclique, créant un effet d'animation de teintes sur le motif de bruit.

Utilisations typiques de `colorama`

- **Créer des effets psychédéliques** : Le balayage rapide des couleurs peut donner un effet vibrant et énergique, idéal pour des créations visuelles immersives.

- **Ajouter du mouvement à des visuels statiques** : Sur des formes ou des motifs fixes, `colorama` permet d'ajouter du dynamisme en jouant uniquement sur la teinte des couleurs, sans modifier les formes ou les positions.

- **Enrichir des textures** : En appliquant `colorama` sur une texture de base (comme un motif de bruit ou un gradient), on peut obtenir des changements subtils et esthétiques qui ajoutent de la profondeur aux visuels.

Avis

`colorama` est un outil puissant dans Hydra pour les artistes visuels qui cherchent à explorer des effets de couleurs changeants sans avoir à modifier la structure sous-jacente de leurs visuels. Il est particulièrement efficace en combinaison avec d'autres fonctions de modulations (comme `modulate` et `osc`) pour créer des animations immersives et captivantes. Utilisé avec modération, `colorama` peut également introduire des nuances de couleurs intéressantes dans des œuvres visuelles abstraites et psychédéliques.

R (G et B)

Dans **Hydra Video Synth**, **R** représente le canal **rouge** (Red) d'une image ou d'une texture. C'est l'une des trois composantes du modèle de couleur **RGB** (Rouge, Vert, Bleu), qui détermine la teinte et l'intensité de la lumière rouge dans le rendu visuel.

Rôle de `R` dans Hydra

1. **Définir l'intensité de la couleur rouge** :

   La valeur de **R** va de **0** à **1**.
   0 : Pas de rouge.
   1 : Rouge à pleine intensité.
   Lorsque vous spécifiez une couleur, **R** contrôle la contribution du rouge à la couleur globale.

Exemple simple :

```
solid(1, 0, 0)

.out()
```

2.
   - **1, 0, 0** signifie : rouge à pleine intensité, pas de vert, pas de bleu.
   - Résultat : l'écran entier sera rouge.

Comment `R` influence les textures dans Hydra

1. **Appliquer une teinte rouge à une texture** :

   Utiliser le paramètre **R** dans la fonction

```
color()
.osc(10, 0.1, 0.8)
.color(1, 0, 0)
.out()
```

- Ici :
  - La texture créée par `osc()` est entièrement teintée de rouge.

2. **Moduler dynamiquement `R`** : Vous pouvez lier la valeur de **R** à des fonctions mathématiques ou des entrées externes (par exemple, le temps ou l'audio) pour changer dynamiquement l'intensité du rouge.

```
osc(10, 0.1, 0.8)

  .color(() => Math.sin(time), 0, 0)
  .out()
```

Math.sin(time) : La valeur de rouge oscille entre 0 et 1 en fonction du temps.

6. **Effets "Split Channel" (Décalage des canaux)** : En isolant ou en manipulant uniquement le canal rouge, vous pouvez créer des effets glitch ou abstraits.

```
osc(10, 0.1, 0.8)

  .color(1, 0, 0) // Rouge pur
  .layer(osc(10, 0.1, 0.8).color(0, 0, 1).scrollX(0.02)) //
Ajout de bleu décalé horizontalement
  .out()
```

7.
   - Ici, le rouge reste fixe, mais un décalage horizontal du bleu crée un effet visuel intéressant.

Utilisation de `R` dans les calculs visuels

Hydra permet de manipuler directement les valeurs des canaux RGB dans les shaders ou les textures :

1. **Extraction du canal rouge d'une texture** :

2.
```
osc(10, 0.1, 0.8).r().out() (à revoir)
```

3.
   - La méthode `.r()` extrait uniquement le canal rouge d'une texture, ce qui donne une image en niveaux de gris basée sur l'intensité du rouge.

3. **Modulation par le rouge** : Vous pouvez utiliser **R** pour moduler d'autres paramètres, comme la vitesse ou l'échelle d'une texture.

```
osc(10, 0.1, 0.8)

  .modulate(noise(3).r(), 0.5)
  .out()
```

- Ici, le bruit est modulé uniquement par l'intensité du rouge.

Résumé

Le canal **R** dans Hydra est essentiel pour :

- **Définir des couleurs** ou créer des teintes dominées par le rouge.
- **Moduler dynamiquement l'intensité du rouge** pour des effets animés ou interactifs.
- **Manipuler directement le canal rouge** pour des effets glitch ou abstraits en vidéo synthèse.
-

BLEND

Dans **Hydra Video Synth**, la fonction `add()` est une méthode permettant de **combiner deux signaux visuels** en additionnant leurs valeurs de pixels correspondants. Ce type d'opération est couramment utilisé en synthèse vidéo pour superposer ou fusionner des formes, des textures, ou des animations.

Fonctionnement de `add()`

Lorsque vous appliquez `add()`, les valeurs de chaque pixel des deux entrées (les textures ou sources visuelles) sont additionnées. Cela peut produire des effets de **luminosité accrue** ou des combinaisons visuelles intéressantes.

Voici quelques points clés sur le comportement de `add()` :

- Les valeurs des pixels sont additionnées **par canal de couleur** (rouge, vert, bleu, alpha).
- Si la somme des valeurs dépasse `1.0` (dans l'échelle normalisée de 0 à 1 utilisée par Hydra), elle est souvent **clippée** à `1.0`, ce qui peut entraîner des zones de couleur saturée ou blanche.
- Les opérations peuvent être affectées par l'**opacité** (canal alpha) des textures combinées.

Syntaxe de `add()`

```
// add(texture2, intensity)
add(src, amount)
```

- **src** : La source visuelle à ajouter à la texture principale.
- **amount** *(optionnel)* : Un multiplicateur qui ajuste l'intensité de la seconde texture avant qu'elle soit ajoutée.

Exemples pratiques

1. Addition simple

```
osc(10, 0.1, 1) // Oscillateur de base
  .add(osc(30, 0.2, 1)) // Ajout d'un oscillateur plus rapide
  .out()
```

Cet exemple superpose deux oscillateurs avec des fréquences différentes, produisant une texture complexe.

2. Avec intensité ajustée

```
osc(20, 0.1, 1)
  .add(osc(40, 0.2, 1), 0.5) // Réduit l'intensité de la
seconde texture
  .out()
```

Le second oscillateur contribue à l'image finale avec seulement 50 % de son intensité.

3. Application sur une vidéo

```
s0.initCam() // Initialisation de la caméra
src(s0)
  .add(osc(15, 0.2, 1)) // Superposition d'une texture
oscillante sur la vidéo
  .out()
```

Cela crée un effet où une texture animée est ajoutée à l'image de la caméra.

Applications créatives

- **Superposition de textures** : Ajoutez des formes, des motifs ou des vidéos pour enrichir une composition.
- **Effets de surbrillance** : Augmentez la luminosité ou créez des éclats visuels.
- **Création de nouveaux motifs** : Combinez plusieurs sources pour générer des motifs uniques.

L'addition est un outil simple mais puissant dans Hydra, permettant de moduler vos créations visuelles et d'explorer une multitude de combinaisons.

SUB

Dans **Hydra Video Synth**, la fonction `sub()` est utilisée pour effectuer une **soustraction pixel par pixel** entre deux signaux visuels. Cela permet de **soustraire la luminosité, les couleurs ou les motifs** d'une texture à une autre, produisant des effets visuels souvent plus sombres ou subtils, comme des **silhouettes inversées** ou des contrastes marqués.

Fonctionnement de `sub()`

- Les valeurs des pixels de la texture principale sont **réduites** par celles de la texture secondaire (source d'entrée).
- La soustraction est effectuée **canal par canal** (rouge, vert, bleu, alpha).
- Si une valeur de pixel devient négative (ce qui peut arriver dans une échelle de 0 à 1), elle est **clippée à zéro**, ce qui donne une couleur noire.

Syntaxe de `sub()`

```
// sub(texture2, intensity)
sub(src, amount)
```

- **src** : La source visuelle à soustraire de la texture principale.
- **amount** *(optionnel)* : Un multiplicateur appliqué à la texture secondaire avant la soustraction.

Exemples d'utilisation

1. Ajustement de l'intensité

```
osc(20, 0.1, 1)
  .sub(osc(40, 0.2, 1), 0.5) // Soustraction avec intensité
réduite de 50 %
  .out()
```

Cela produit un effet plus subtil, où la soustraction est moins dominante.

2. Effet de silhouette avec une vidéo

```
s0.initCam() // Initialisation de la caméra
src(s0)
  .sub(osc(15, 0.2, 1).0.5) // Soustraction d'une texture
oscillante sur la vidéo
  .out()
```

Cet effet donne l'impression que les motifs de l'oscillateur "effacent" certaines parties de la vidéo, créant des formes sombres dynamiques.

Applications créatives

1. **Création de contrastes dramatiques** :
    - Utilisez `sub()` pour révéler ou masquer certaines parties d'une texture en fonction d'une source secondaire.
2. **Effets de silhouette inversée** :

- Soustraire une texture lumineuse d'une source sombre peut créer des effets visuels ressemblant à des ombres inversées ou à des motifs gravés.

3. **Interaction entre formes et textures** :

   - Combinez des formes complexes en soustrayant des textures dynamiques (comme des oscillateurs) pour des motifs plus riches.

Comparaison avec `add()`

Aspect	add()	sub()
Effet principal	Rend l'image plus lumineuse	Rend l'image plus sombre
Fonction	Ajoute les valeurs des pixels	Soustrait les valeurs des pixels
Applications	Superpositions, éclats lumineux	Silhouettes, ombres, contrastes

La fonction `sub()` est un outil puissant pour explorer des variations sombres et abstraites dans vos compositions Hydra. Elle permet de sculpter visuellement vos textures en jouant sur les contrastes et les absences.

LAYER

(A AMELIORER)

Dans **Hydra Video Synth**, une *layer* (ou couche) est un élément de base qui permet de construire des compositions visuelles en empilant et en combinant différentes sources graphiques. Ces couches interagissent entre elles pour produire des effets complexes et dynamiques. Chaque *layer* peut représenter un flux vidéo ou une texture générée en temps réel.

Comprendre les Layers dans Hydra

1. **Concept de Base : Empilement Visuel**
   Une *layer* agit comme une feuille transparente où vous pouvez dessiner, appliquer des effets ou combiner des sources. Les couches s'empilent pour créer une composition finale, chaque couche pouvant interagir avec les autres.
   Pensez aux *layers* comme des calques dans un logiciel de design : ils peuvent être manipulés individuellement ou fusionnés.

2. **Principaux Outils pour Gérer les Layers**
   Hydra propose des commandes simples pour travailler avec les couches :

   - **`layer(n)`** : sélectionne une couche spécifique, où `n` est l'indice (0 pour la première couche, 1 pour la suivante, etc.).
   - **`blend()`** : fusionne deux couches avec une transparence.
   - **`modulate()`** : utilise une couche comme masque ou modulation pour une autre.

- **`out()`** : rend visible la sortie d'une ou plusieurs couches.

Exemples de Composition avec Layers

Exemple 1 : Superposition avec Transparence

On peut superposer des oscillateurs avec différents réglages pour produire des motifs riches.

```
//  Un oscillateur bleu en mouvement

osc(5, 0.2, 1).layer(osc(0.5, 0, 1)).out()
```

Exemple 2 : Effet de Masque avec Modulation

Les couches peuvent interagir comme des masques pour découper ou moduler les textures.

```
// solid animé

solid([1,0,0],[0,1,0],
[0,0,1],1).layer(noise().rotate(1).luma()).out(o0)
```

Exemple 3 : Feedback Visuel

Les layers permettent de créer des effets de rétroaction (feedback), où la sortie d'une couche est réinjectée comme source.

EXEMPLE S A REVOIR

```
// Couche 0 : Oscillateur de base
layer(0).src(osc(20, 0.1).rotate(0.3)).out()

// Couche 1 : Ajout de feedback visuel
layer(1).src(o0).scale(0.9).rotate(0.1).blend(layer(0),
0.7).out()
```

Exemple 4 : Fusion Vidéo

Les *layers* permettent aussi de mixer des vidéos, comme une webcam ou une vidéo préenregistrée.

```
s0.initVideo("https://media.giphy.com/media/3o7abldj0b3rxrZUxW/
giphy.mp4")

src(s0).

layer(

  src(o0)

    .scale(0.9)

    .rotate(0.1)

    .blend(src(s0), 0.5))

.out()
```

BLEND

Dans **Hydra Video Synth**, la fonction **`blend`** permet de combiner deux sources visuelles (ou couches) en ajustant leur opacité relative. Elle crée une interpolation linéaire entre deux textures ou vidéos en fonction d'un paramètre donné.

Fonctionnement de `blend`

La syntaxe générale est la suivante :

```
blend(src, amount)
```

- **`src`** : La source visuelle ou la couche que vous voulez mélanger avec la couche actuelle.
- **`amt`** : Un paramètre entre `0` et `1` qui contrôle le niveau de mélange.
  - `0` signifie que seule la couche actuelle est visible (aucune fusion).
  - `1` signifie que seule la nouvelle source (src) est visible.
  - Les valeurs intermédiaires (par exemple, `0.5`) créent une fusion équilibrée entre les deux couches.

Exemple d'utilisation simple

```
osc(10, 0.1, 0.5) // Génère un oscillateur
  .blend(osc(20, 0.2, 0.5), 0.5) // Mélange avec une seconde
source
  .out()
```

Dans cet exemple :

- Un oscillateur à basse fréquence est généré en arrière-plan.
- Un autre oscillateur, avec une fréquence différente, est fusionné à 50 % avec le premier.

Effets dynamiques avec `blend`

Pour des effets plus complexes, vous pouvez animer le paramètre **`amt`** en utilisant une fonction comme `time` ou `modulate`.

Exemple :

```
osc(10, 0.1, 0.5)
  .blend(osc(20, 0.2, 0.5), Math.sin(time) * 0.5 + 2) //
Fusion dynamique basée sur le temps
  .out()
```

Ici, le paramètre `amt` oscille grâce à une fonction sinus, créant une transition fluide entre les deux couches au fil du temps.

Applications créatives

- **Superposition subtile** : Ajouter de la texture ou des motifs sur une vidéo ou une animation.
- **Transitions douces** : Passer progressivement d'une source à une autre.
- **Effets glitch ou psychédéliques** : Mélanger des formes, des couleurs ou des motifs qui se déplacent dynamiquement.

En résumé, la fonction **`blend`** est un outil puissant dans Hydra pour créer des visuels dynamiques et variés en combinant des sources avec précision.

MULT

Fonctionnement de `mult`

La syntaxe :

`mult(src, amt)`

- **`src`** : La source visuelle ou la couche avec laquelle multiplier.
- **`amt`** *(facultatif)* : Un facteur d'intensité entre `0` et `1` (ou plus) qui contrôle à quel point la multiplication affecte la couche actuelle.
  - `0` : Aucune multiplication (la couche actuelle reste inchangée).
  - `1` : Multiplication normale (la valeur par défaut).
  - Une valeur supérieure à `1` amplifie l'effet.

Multiplication contrôlée avec `amt`

Exemple simple avec un `amt` :

```
osc(10, 0.1, 0.5) // Oscillateur de base
  .mult(shape(4, 0.5), 0.5) // Multiplie avec une forme avec
un effet réduit (50%)
  .out()
```

Dans cet exemple :

- L'oscillateur est combiné avec une forme en utilisant une intensité de 0,5.
- Cela réduit l'effet de la multiplication, rendant la forme moins dominante.

Exemple avec une animation dynamique de `amt` :

```
osc(10, 0.1, 0.5)
  .mult(shape(3, 0.5), Math.sin(time)) // `amt` oscille entre
-1 et 1
  .out()
```

- Ici, l'intensité du mélange est modulée dynamiquement par une fonction sinus, créant une pulsation.

Comparaison entre `mult` et `blend`

- **`mult`** multiplie pixel par pixel, souvent utilisé pour **masquer** ou moduler des couches visuelles.
- **`blend`** effectue une interpolation linéaire, utile pour **superposer** ou mélanger deux sources de manière douce.

Applications créatives de `mult` avec `amt`

1. **Effet de masque dynamique** :

2. ```
   osc(20, 0.1, 0.8)
   ```

3. ```
     .mult(noise(3, 0.1), 0.7) // Contrôle de la force du masquage
   ```

4. ```
     .out()
   ```

5.

6. **Transitions complexes** :

7. ```
   osc(15, 0.2, 0.5)
   ```

8. ```
     .mult(osc(5).rotate(0.5), Math.sin(time) * 0.5 + 0.5) // Mélange avec une source en transition fluide
   ```

9. ```
     .out()
   ```

10.

11. **Amplification visuelle** : Utilisez des valeurs de `amt` supérieures à 1 pour intensifier certaines zones :

12. ```
    osc(10, 0.1, 0.5)
    ```

13. ```
      .mult(noise(3, 0.2), 1.5) // Amplifie les motifs générés par le bruit
    ```

14. ```
      .out()
    ```

15.

Conclusion

La fonction **`mult(src, amt)`** est un outil puissant dans Hydra pour contrôler la manière dont les couches interagissent visuellement. Le paramètre **`amt`** ajoute une flexibilité supplémentaire, permettant d'ajuster finement l'impact de la multiplication, que ce soit pour créer des masques subtils ou des effets visuels intenses.
    ```

DIFF

Dans **Hydra Video Synth**, l'opérateur `diff` est utilisé pour calculer la différence absolue entre deux couches (ou sources visuelles). Il s'agit d'une méthode pour créer des effets visuels où les différences entre deux images ou flux sont mises en évidence, souvent avec des résultats très dynamiques ou texturés.

Utilisation dans Hydra

Voici un exemple typique d'utilisation de `diff` dans un patch Hydra :

```
osc(10, 0.1, 1)
  .diff(osc(15, 0.05, 2).rotate(0.5))
  .out()
```

Explication :

1. La première couche est un oscilloscope (`osc`) avec une fréquence de 10, une vitesse de modulation de 0.1 et une intensité de 1.
2. La deuxième couche est un autre `osc` avec une fréquence de 15, une modulation de 0.05, une intensité de 2, et une rotation appliquée de 0.5.
3. Le `diff` prend ces deux couches et calcule leur différence absolue pour créer un effet contrasté entre elles.

Effets typiques du `diff`

- **Textures dynamiques :** Les couches oscillantes ou en mouvement produisent des motifs vivants et en constante évolution.
- **Effets de moiré :** Avec des fréquences similaires entre deux oscillateurs, `diff` peut révéler des motifs d'interférences complexes.
- **Effets de contraste :** Il est utile pour mettre en évidence les zones de divergence entre deux flux visuels.

Exploration créative

- Combinez `diff` avec d'autres opérateurs comme `.add()`, `.sub()`, ou `.modulate()` pour générer des effets plus complexes.
- Appliquez-le à des sources vidéo ou caméras pour transformer des scènes en des motifs abstraits.
- Associez-le à des paramètres dynamiques (`.scale()`, `.rotate()`, `.kaleid()`) pour expérimenter avec des compositions évolutives.

Exemple avancé avec une webcam :

```
s0.initCam()
src(s0)
  .diff(osc(20, 0.1, 0.5).rotate(0.2))
  .out()
```

Cela crée une interaction entre l'entrée de la webcam et un oscillateur, produisant un rendu abstrait et organique.

En résumé, `diff` est un opérateur puissant pour jouer avec les contrastes visuels et révéler des détails ou des motifs dans vos compositions vidéo génératives.

MASK

Dans **Hydra Video Synth**, la fonction `mask()` est utilisée pour appliquer un masque sur une source visuelle. Un masque est une technique permettant de filtrer ou masquer certaines parties d'une image ou d'un flux vidéo en fonction d'une texture ou d'un critère défini. C'est une opération courante en traitement d'image pour créer des effets visuels dynamiques ou complexes.

Fonctionnement de `mask()`

Le masque fonctionne en prenant une **texture** (source visuelle) et en utilisant une **autre texture** ou un **seuil** comme "gabarit" pour définir quelles parties de l'image source seront visibles.

Syntaxe de base :

```
src(source).mask(texture, scale, offset) (??)
```

Paramètres :

1. **source** : la source vidéo ou l'image sur laquelle appliquer le masque.
2. **texture** : la texture utilisée comme masque. Les parties claires de cette texture laisseront passer l'image d'origine, tandis que les parties sombres masqueront l'image.
3. **scale** (optionnel) : facteur d'échelle pour ajuster la taille du masque par rapport à la source.
4. **offset** (optionnel) : décalage (x, y) pour ajuster la position du masque.

Exemple simple :

```
// Création d'une source de base

gradient(5)
  .mask(osc(10, 0.1, 1)) // Applique un masque basé sur un
oscillateur
  .out()
```

Dans cet exemple :

- La source `o0` est affichée, mais seule une partie sera visible.
- Le masque est généré par `osc(10, 0.1, 1)`, une texture oscillante.
-

Exemple avancé avec échelle et décalage :

```
osc(10)
  .mask(shape(4, 0.5, 0.3), [1.5, 1.5], [0.5, 0.5])
  .out() (??)
```

Dans cet exemple :

- Le masque est basé sur une forme (`shape`).
- L'échelle est ajustée avec `[1.5, 1.5]` pour rendre le masque plus grand.
- L'offset `[0.5, 0.5]` décale la position du masque.(?)
-

Cas d'usage :

1. **Création d'effets dynamiques** : Limiter les animations à certaines zones.
2. **Superposition artistique** : Combiner plusieurs sources visuelles avec des masques.
3. **Effets de transition** : Introduire ou faire disparaître progressivement des éléments visuels.

Le **mask** dans Hydra est particulièrement utile pour expérimenter et créer des visuels complexes dans des performances live ou des installations artistiques.

MODULATE

MODULATEREPEAT

Dans **Hydra Video Synth**, la fonction **`modulateRepeat`** est un opérateur qui applique une modulation répétitive à une texture d'entrée en utilisant une texture de modulation. Cette fonction est utilisée pour créer des effets complexes et répétitifs, souvent liés à des motifs ou des distorsions spatiales dynamiques.

Voici une explication détaillée des paramètres et de son fonctionnement :

Syntaxe

```
modulateRepeat(src, repeatX, repeatY, offsetX, offsetY)
```

Paramètres

1. **`src`** : La texture de modulation.
    - C'est la source qui sera utilisée pour moduler l'image ou la texture actuelle.
    - Par exemple, vous pouvez utiliser un oscillateur (`osc()`) ou toute autre texture.
2. **`repeatX`** : Fréquence de répétition horizontale.
    - Définit combien de fois la texture sera répétée sur l'axe horizontal.
    - Une valeur plus grande entraîne des répétitions plus fréquentes.
3. **`repeatY`** : Fréquence de répétition verticale.
    - Définit combien de fois la texture sera répétée sur l'axe vertical.
4. **`offsetX`** (optionnel) : Décalage horizontal.
    - Permet de décaler les répétitions horizontalement pour un effet dynamique.
5. **`offsetY`** (optionnel) : Décalage vertical.
    - Permet de décaler les répétitions verticalement.

Fonctionnement

- Cette fonction applique un motif répétitif basé sur les coordonnées générées par la texture de modulation (`src`).
- Les paramètres de répétition permettent de diviser l'écran en une grille définie par `repeatX` et `repeatY`.
- L'effet est particulièrement utile pour créer des symétries, des mosaïques, ou des structures géométriques modulées par une source dynamique.

Exemple d'utilisation

```
shape(4,0.9)
  .add(osc(3,0.5,1))
  .modulateRepeat(osc(10), 3.0, 5.0, 0.5, 0.5)
  .out(o0)
```

- **`shape(4,0.9)`** génère un carré.
- **`.add(osc(3,0.5,1))`** est utilisé comme texture de modulation.
- **`3, 5`** signifie que la texture est répétée trois et 5 fois sur chaque axe (x et y).
- **`0.5, 0.5`** ajoute un léger décalage pour dynamiser l'effet.

Applications courantes

1. **Création de motifs répétitifs** : Pour des effets visuels inspirés des mosaïques ou de l'art génératif.
2. **Synchronisation avec l'audio** : En utilisant des textures dynamiques basées sur l'entrée sonore.
3. **Exploration géométrique** : Jouer avec des valeurs dynamiques pour expérimenter des effets abstraits.

MODULATEREPEATX

Dans **Hydra Video Synth**, la fonction `modulateRepeatX` est utilisée pour manipuler une texture en la répétant le long de l'axe X (horizontal) et en modulant cette répétition en fonction d'une autre texture ou source. Cela crée des effets visuels dynamiques et complexes, souvent utilisés pour générer des motifs répétitifs avec des distorsions intéressantes.

Syntaxe

```
modulateRepeatX(source, reps = 3, offset = 0.5)
```

Paramètres

1. **`source`**
   La texture ou source utilisée pour moduler la répétition. Cela peut être une autre sortie vidéo ou un générateur dans Hydra (comme `osc`, `shape`, etc.).

2. **`Reps`**
   Contrôle le nombre de répétitions par défaut à 3

3. **`offset`** *(par défaut : 0.5)*
   Définit le décalage de la modulation, influençant la façon dont le motif est appliqué à la texture.

Fonctionnement

- **Répétition Horizontale** : `modulateRepeatX` commence par diviser l'image ou la texture en plusieurs sections répétées le long de l'axe X.
- **Modulation Dynamique** : Ensuite, la source spécifiée (premier paramètre) est utilisée pour déformer ou perturber ces répétitions en fonction de ses propres variations.
-

Exemple Pratique

Voici un exemple simple pour visualiser l'effet de `modulateRepeatX` :

```
noise(2)
  .modulateRepeatX(gradient(), 30,0.9)
.colorama(2)
.out()
```

- **`noise(2)`** génère une texture oscillante.
- **`modulateRepeatX(gradient(), 30, 0.9)`** utilise une autre oscillation (`gradient()`) pour moduler les répétitions de la première texture.

- Le résultat est une série de motifs horizontaux, avec des variations dynamiques selon l'oscillation modulateur.
-

Effets Visuels

- Cela peut produire des effets de type "ondes", des déformations fluides ou des distorsions géométriques selon la source utilisée.
- Idéal pour ajouter des perturbations texturales dans des compositions visuelles en temps réel.

MODULATEREPEATY

Dans **Hydra Video Synth**, la fonction `modulateRepeatY` est utilisée pour manipuler une texture en la répétant le long de l'axe Y (horizontal) et en modulant cette répétition en fonction d'une autre texture ou source. Cela crée des effets visuels dynamiques et complexes, souvent utilisés pour générer des motifs répétitifs avec des distorsions intéressantes.

Syntaxe

```
modulateRepeatY(source, reps = 3, offset = 0.5)
```

1. **`source`**
   La texture ou source utilisée pour moduler la répétition. Cela peut être une autre sortie vidéo ou un générateur dans Hydra (comme `osc`, `shape`, etc.).

2. **`Reps`**
   Contrôle le nombre de répétitions par défaut à 3

3. **`offset`** *(par défaut : 0.5)*
   Définit le décalage de la modulation, influençant la façon dont le motif est appliqué à la texture.

Fonctionnement

- **Répétition Horizontale** : `modulateRepeatX` commence par diviser l'image ou la texture en plusieurs sections répétées le long de l'axe Y.
- **Modulation Dynamique** : Ensuite, la source spécifiée (premier paramètre) est utilisée pour déformer ou perturber ces répétitions en fonction de ses propres variations.
-

Exemple Pratique

Voici un exemple simple pour visualiser l'effet de `modulateRepeatX` :

```
gradient(2)
  .modulateRepeatY(gradient(), 30,0.9)
.colorama(2)
.saturate(20)
.out()
```

- **`gradient(2)`** génère une texture oscillante.
- **`modulateRepeatY(gradient(), 30, 0.9)`** utilise une autre oscillation (`gradient()`) pour moduler les répétitions de la première texture.
- Le résultat est une série de motifs horizontaux, avec des variations dynamiques selon l'oscillation modulateur.

MODULATEKALEID

Dans Hydra, la fonction `modulateKaleid` est un effet de modulation visuelle qui applique une transformation en kaléidoscope à un flux vidéo ou à une texture en fonction d'un autre flux ou texture utilisé comme modulateur. Cet effet crée des motifs symétriques en déformant l'image originale à l'aide des propriétés du kaléidoscope. Cela permet d'ajouter des éléments de symétrie et des variations dynamiques dans une composition visuelle.

Voici une explication détaillée de ses paramètres et de son fonctionnement :

Syntaxe

```
modulateKaleid(texture, nSides)
```

Paramètres

1. **`input`** : La source ou texture utilisée pour moduler l'effet kaléidoscope. Cela peut être une texture générée dans Hydra, une vidéo ou une image.
2. **`nSides`** : (Nombre de côtés) Définit le nombre de symétries dans le kaléidoscope. Par exemple :
   - `3` crée un effet triangulaire.
   - `6` génère un effet hexagonal.
   - Une valeur élevée ajoute davantage de symétries.

Exemple simple d'utilisation

```
osc(10, 0.1, 1)
  .modulateKaleid(osc(5, 0.2, 1), 6)
  .out()
```

Analyse du code

1. **`osc(10, 0.1, 1)`** : Génère une texture d'oscillation.

2. **`modulateKaleid(osc(5, 0.2, 1), 6, 0.5)`** :
    - Utilise un autre oscillateur (`osc(5, 0.2, 1)`) comme source de modulation.
    - Applique une symétrie avec 6 côtés (`nSides = 6`).

3. **`.out()`** : Envoie la composition visuelle en sortie.

Effets et applications

- **Création de motifs dynamiques** : Idéal pour générer des animations psychédéliques.
- **Effets réactifs** : Peut être combiné avec d'autres sources (comme de l'audio) pour des effets synchronisés.
- **Complexité visuelle** : En manipulant les paramètres dynamiquement (par des fonctions ou des contrôleurs), vous pouvez obtenir des compositions captivantes.

Exemple avec interaction dynamique

```
voronoi(10, 5)
  .modulateKaleid(osc(10, 0.2, 0.5), Math.sin(time) * 10 + 3)
  .out()
```

Ici, le nombre de côtés et l'intensité changent au fil du temps grâce à la fonction sinus.

Conclusion

`modulateKaleid` est un outil puissant pour ajouter de la symétrie et de la profondeur visuelle dans les compositions Hydra. En jouant avec les paramètres et en utilisant différentes textures d'entrée, il permet de créer une grande variété d'effets uniques.

MODULATESCROLLX(MODULATESCROLLY)

Dans **Hydra**, la fonction `modulateScrollX` est un effet visuel qui déforme une texture ou une source visuelle en appliquant une modulation sur le défilement horizontal (**ScrollX**), en fonction d'une texture ou d'une source modulateur. Cet effet permet de créer des mouvements fluides, des distorsions dynamiques ou des effets ondulatoires dans l'axe horizontal.

Syntaxe

```
modulateScrollX(texture, scrollX = 0.5, speed)
```

Paramètres

1. **`input`** :
    - La source ou texture utilisée pour moduler le défilement.

- Cela peut être une texture générée dans Hydra (comme `osc`, `noise`, etc.) ou une vidéo/image.

2. **`scrollX`** : Intensité de modulation par défaut à 0.5

3. **`Speed`**: vitesse de défilement

Exemple de base

```
osc(10, 0.1, 1) // Génère une oscillation visuelle.
  .modulateScrollX(osc(5, 0.2, 1), 0.5, 0.2) // Applique une
modulation horizontale.
  .out() // Envoie le résultat en sortie.
```

Explication

1. **`osc(10, 0.1, 1)`** : Texture d'oscillation avec des lignes.
2. **`modulateScrollX(osc(5, 0.2, 1), 0.5, 0.2)`** :
   - Utilise un autre oscillateur comme modulateur.
   - Intensité de modulation réglée à 0.5.
   - Défilement horizontal à une vitesse de 0.2.

Effets et combinaisons dynamiques

Vous pouvez obtenir des effets visuels captivants en combinant `modulateScrollX` avec d'autres modulateurs ou en changeant les paramètres dynamiquement :

Exemple avec changement dynamique :

```
gradient(0.2)
  .modulateScrollX(noise(3, 0.5), 0.8, Math.sin(time) * 0.3)
  .modulate(noise(2, 0.1), 0.2)
  .out()
```

Analyse :

- **`Math.sin(time)`** et **`Math.cos(time)`** permettent de varier l'intensité et la vitesse au fil du temps.
- Résultat : un mouvement horizontal qui fluctue selon des cycles sinusoïdaux.

Applications créatives

1. **Ondulations horizontales dynamiques** : Utilisez une source comme `noise` ou `gradient` pour ajouter des vagues ou des distorsions.
2. **Effets réactifs** : Synchronisez le paramètre `scrollX` ou `amount` avec de la musique ou d'autres signaux.
3. **Transitions abstraites** : Combinez avec `blend`, `rotate`, ou d'autres modulations pour obtenir des visuels complexes.

Résumé

`modulateScrollX` et donc `modulateScrollY` est une fonction essentielle pour ajouter des mouvements horizontaux subtils ou dramatiques dans vos compositions Hydra. En jouant avec les paramètres et en utilisant différentes sources de modulation, elle permet de générer des animations fluides, hypnotiques ou chaotiques selon vos besoins artistiques.

MODULATE

Dans **Hydra Video Synth**, la fonction **modulate** est utilisée pour modifier ou perturber une texture ou un signal visuel en fonction d'une autre texture ou signal. Cela crée des effets dynamiques et interactifs, souvent utilisés pour générer des visuels complexes et fluides. Elle agit comme une modulation, c'est-à-dire une manière de combiner deux entrées (une source et une texture modulatrice) pour produire un résultat visuel unique.

Fonctionnement de modulate

La commande **modulate** fonctionne en superposant ou en perturbant une texture d'entrée à l'aide d'une seconde texture. Cela peut inclure des translations, des rotations, ou des déformations en fonction de la valeur des pixels de la texture modulatrice.

La syntaxe générale est la suivante :

```
src(o0) // Source originale
  .modulate(src(o1), amount, offset)
  .out(o0) // Sortie visuelle
```

Paramètres :

1. **texture** : La texture ou le signal utilisé pour moduler (par exemple, une autre sortie, une forme ou une vidéo).
   Exemple : `src(o1)`, `shape(4)`, etc.

2. **amount** : Un nombre (ou une fonction) entre 0 et 1, qui détermine l'intensité de la modulation. Une valeur proche de 1 signifie que la modulation aura un effet très visible, tandis qu'une valeur proche de 0 atténue son impact.

3. **offset** (optionnel) : Décale la texture utilisée pour la modulation. Cela peut ajouter des variations supplémentaires et enrichir les visuels.

Exemple d'utilisation :

Exemple basique :

```
osc(10, 0.1, 0.8) // Oscillateur en entrée
  .modulate(osc(20, 0.2), 0.5) // Modulation avec un autre
oscillateur
  .out(o0)
```

Ici, un oscillateur à une fréquence de 20 modifie l'oscillateur de base (fréquence 10) avec une intensité de 0.5.

Exemple avec une forme :

```
shape(4, 0.5, 0.1) // Un carré
  .modulate(noise(3), 0.8) // Modulé par un bruit
  .out(o0)
```

Dans cet exemple, une forme carrée est perturbée par un signal de bruit, créant un effet de déformation fluide.

Variations de modulate :

Hydra propose plusieurs variantes de modulate pour des usages spécifiques :

- **modulateScale** : Modifie l'échelle en fonction d'une texture.
- **modulateRotate** : Modifie la rotation selon une texture.
- **modulatePixelate** : Pixelise une texture en fonction d'une autre.
- **modulateHue** : Modifie la teinte (couleur) en fonction d'une autre texture.

Exemple avec modulateScale :

```
osc(10, 0.1, 0.8)
  .modulateScale(osc(20, 0.2), 0.5)
  .out(o0)
```

Cela crée un effet de zoom basé sur une seconde texture.

Résumé :

La fonction **modulate** est un outil puissant pour enrichir vos visuels dans Hydra. En jouant sur les textures, les fréquences, et les paramètres, vous pouvez créer des effets visuels dynamiques et organiques. C'est un élément central pour ceux qui explorent des visuels génératifs et veulent ajouter des interactions entre leurs sources.

MODULATESCALE

Dans **Hydra Video Synth**, la fonction `modulateScale` est utilisée pour moduler l'échelle (la taille ou le zoom) d'une texture en fonction d'une autre texture. Cela permet de créer des effets visuels dynamiques où une texture agit sur l'échelle d'une autre, souvent dans des compositions interactives et génératives.

Fonctionnement de `modulateScale`

```
modulateScale(texture, multiple = 1, offset = 0)
```

Paramètres

1. **`texture`** : La texture ou la source utilisée pour moduler l'échelle. Cela peut être un flux vidéo, une oscillation, du bruit ou une autre forme générée dans Hydra.
2. **`multiple(amount)`** (par défaut : `1`) : La force de la modulation appliquée. Une valeur positive augmente l'effet, tandis qu'une valeur proche de zéro le réduit.
3. **`offset`** (par défaut : `0`) : Un décalage appliqué à la texture modulatrice, permettant de contrôler l'effet à un niveau plus subtil.

Résultat

La texture principale est agrandie ou réduite en fonction des valeurs de luminosité ou d'intensité de la texture modulatrice. Par exemple :

- Les zones plus claires de la texture modulatrice peuvent provoquer un zoom avant.
- Les zones plus sombres peuvent provoquer un zoom arrière.

Exemple d'utilisation

Voici un exemple d'effet visuel avec `modulateScale` :

```
osc(10, 0.1, 1)
  .modulateScale(noise(5),0.2,0.9)
  .out()x
```

Dans cet exemple :

- Une oscillation rapide (`osc(10, 0.1, 1)`) est créée comme texture principale.
- Une oscillation plus lente (`osc(5)`) module l'échelle de la première texture, générant un effet de pulsation ou de vibration.

Cas pratique

`modulateScale` est particulièrement utile pour :

- Créer des animations organiques où les textures semblent "respirer" ou "pulsent".
- Ajouter de la complexité visuelle en combinant des sources dynamiques.

- Générer des effets réactifs à la musique ou à d'autres stimuli si Hydra est synchronisé avec des données externes.

Tu peux expérimenter avec des sources plus complexes (comme `noise`, `shape`, ou des vidéos) pour voir comment elles influencent la modulation.

MODULATEPIXELATE

Vous avez raison, les paramètres de la fonction `modulatePixelate` dans **Hydra Video Synth** sont différents de ceux mentionnés plus tôt. Voici une description corrigée et détaillée de la fonction :

Fonction : `modulatePixelate`

La fonction `modulatePixelate` applique une modulation pixelisée en fonction d'une texture de source. Elle combine les valeurs de la texture d'origine et de la texture source pour produire un effet où la taille et la structure des pixels changent dynamiquement.

Syntaxe

```
.modulatePixelate(texture, multiple, offset)
```

Paramètres

1. **texture** :
    - La texture utilisée comme source de modulation. Cela peut être une texture générée (comme `osc()`, `voronoi()`, `noise()`) ou une entrée vidéo/caméra (comme `s0`, `s1`).
    - Cette texture influence la distribution et la dynamique de la pixelisation.
2. **multiple** :
    - Définit le **facteur multiplicateur** pour la taille des pixels.
    - Plus cette valeur est grande, plus les pixels deviennent petits, produisant un effet de densité accrue.
3. **offset** :
    - Définit un décalage (offset) appliqué à l'effet de pixelisation.
    - Cela permet de contrôler la variation et la distribution des zones affectées par la pixelisation.

Exemple d'utilisation simple

```
osc(30, 0.1, 1)
  .modulatePixelate(noise(5), 100, 0.1)
  .out()
```

Décomposition :

- **Texture de base** : `osc(30, 0.1, 1)` génère une texture oscillante avec une fréquence de 30.
- **Texture de modulation** : `noise(5)` crée une texture bruitée utilisée pour moduler la pixelisation.
- **Paramètres de modulation** :
  - `multiple = 10` contrôle la taille des pixels, les rendant assez petits.
  - `offset = 0.1` ajoute une légère variation au positionnement de la pixelisation.

Exemple avancé

```
 speed = 0.2
voronoi(15, 0.3, 0.1)
  .modulatePixelate(osc(10, 0.2, 0.8), 50, 0.2)
  .out()
```

Décomposition :

- **Texture de base** : `voronoi(15, 0.3, 0.5)` crée un motif en mosaïque basé sur un diagramme de Voronoï.
- **Texture de modulation** : `osc(10, 0.2, 0.8)` est une oscillation lente utilisée pour moduler la taille des pixels.
- **Paramètres de modulation** :
  - `multiple = 50` crée des pixels d'une taille moyenne.
  - `offset = 0.2` ajoute une variation subtile pour dynamiser l'effet.

Applications créatives

1. **Esthétique rétro et pixélisée** : recréez des visuels similaires aux anciens écrans basse résolution.
2. **Visualisations dynamiques** : combinez `modulatePixelate` avec des textures interactives ou réactives (par exemple, audio ou vidéo).
3. **Effet de chaos structuré** : en modulant avec des textures comme `noise()` ou des entrées caméras en mouvement.

MODULATEROTATE

Vous avez raison, dans **Hydra Video Synth**, les paramètres de la fonction `modulateRotate` sont **texture**, **multiple**, et **offset**. Voici une description précise et complète de cette fonction.

Fonction : `modulateRotate`

La fonction `modulateRotate` applique une modulation dynamique de rotation à une texture, basée sur une autre texture source. Cela crée un effet où l'angle de rotation varie en fonction des valeurs de la texture modulante, offrant des visuels dynamiques et souvent hypnotiques.

Syntaxe

```
.modulateRotate(texture, multiple, offset)
```

Paramètres

1. **texture** :
    - La texture utilisée pour moduler la rotation.
    - Cela peut être une texture générée (`osc()`, `noise()`, `voronoi()`) ou une entrée externe (`s0`, `s1`).
    - Cette texture détermine la manière dont la rotation est modulée spatialement et temporellement.
2. **multiple** :
    - Contrôle la **multiplication de la modulation de rotation**.
    - Plus la valeur est élevée, plus les variations angulaires sont amplifiées.
    - Des valeurs positives modifient la rotation dans le sens horaire, et des valeurs négatives dans le sens antihoraire.
3. **offset** :
    - Ajoute un **décalage constant** à l'angle de rotation.
    - Ce décalage influence la position de départ ou la base de la rotation.

Exemple d'utilisation simple

```
osc(30, 0.1, 1)
  .modulateRotate(noise(4), 1.5, 0.2)
  .out()
```

Décomposition :

- **Texture de base** : `osc(30, 0.1, 1)` génère une onde oscillante avec une fréquence de 30.

- **Texture de modulation** : `noise(4)` fournit une texture bruitée pour moduler la rotation.
- **Paramètres de modulation** :
  - `multiple = 1.5` amplifie la rotation par un facteur modéré.
  - `offset = 0.2` ajoute un léger décalage de base à la rotation.

Résultat : L'onde oscillante est déformée par une rotation dynamique influencée par le bruit, avec un mouvement fluide et légèrement décalé.

Exemple avancé

```
voronoi(20, 0.3, 0.5)
  .modulateRotate(osc(10, 0.2, 0.8), -2, 0.1)
  .out()
```

Décomposition :

- **Texture de base** : `voronoi(20, 0.3, 0.5)` génère un motif en mosaïque basé sur un diagramme de Voronoï.
- **Texture de modulation** : `osc(10, 0.2, 0.8)` fournit une oscillation lente utilisée pour moduler la rotation.
- **Paramètres de modulation** :
  - `multiple = -2` applique une rotation amplifiée dans le sens antihoraire.
  - `offset = 0.1` ajoute un décalage subtil pour des variations dynamiques.

Résultat : Le motif Voronoï tourne avec des angles modulés par l'oscillation, créant un mouvement hypnotique.

Applications créatives

1. **Effet kaléidoscopique** : Combinez `modulateRotate` avec `kaleid()` pour créer des motifs symétriques en rotation.
2. **Visualisations réactives** : Utilisez des textures audio-réactives pour moduler la rotation de manière synchronisée avec la musique.
3. **Distorsions fluides** : Associez des textures complexes comme `noise()` ou `voronoi()` pour obtenir des effets abstraits et dynamiques.

MODULATEHUE

Dans **Hydra**, la fonction `modulateHue` modifie dynamiquement la teinte (hue) d'une texture ou d'un flux vidéo en fonction des variations de teinte d'une autre texture. Elle agit en manipulant le canal de teinte dans le modèle de couleur HSV, créant des effets de décalage ou de mélange de couleurs sophistiqués. Voici une explication plus approfondie, suivie d'exemples précis.

Syntaxe complète :

```
modulateHue(texture, amount, offset)
```

Paramètres détaillés :

1. **`source`** *(texture)* :
    - La texture ou l'image utilisée comme modulateur.
    - Elle influence la manière dont la teinte de la texture principale sera modifiée.
2. **`amount`** *(nombre décimal)* :
    - L'intensité de l'effet de modulation.
    - Une valeur positive renforce le changement de teinte, tandis qu'une valeur négative inverse les décalages.
3. **`offset`** *(nombre décimal, optionnel)* :
    - Décale les valeurs de teinte de la texture principale avant d'appliquer la modulation.
    - Cela ajoute une variation supplémentaire pour enrichir l'effet visuel.

Fonctionnement détaillé :

`modulateHue` ne change pas directement la texture principale, mais superpose une modification en fonction des variations de teinte dans la texture modulateur. Contrairement à d'autres fonctions comme `modulate` (qui agit sur la luminosité), `modulateHue` s'applique uniquement au canal de teinte, ce qui conserve l'intensité et la saturation des couleurs d'origine tout en créant un effet de rotation chromatique.

Exemples d'utilisation :

TOUS LES EXEMPLES SONT A REVOIR!

1. Modulation simple de teinte :

```
osc(20, 0.1, 1) // Une onde oscillante rapide
  .modulateHue(osc(10, 0.05, 0), 0.5) // Modulation avec une
onde plus lente
  .out()
```

- **Description** : La première onde est modulée par une seconde onde plus lente. Cela crée un effet où les teintes se décalent et oscillent dynamiquement dans le spectre colorimétrique.

2. Ajout d'un décalage de teinte (`offset`) :

```
gradient(1) // Génère un dégradé de base
```

```
  .modulateHue(osc(10, 0.1, 0), 0.7, 0.5) // Modulation avec
décalage de teinte
  .out()
```

- **Description** : Le `offset` de `0.5` ajoute un décalage fixe aux teintes avant modulation, ce qui entraîne une rotation constante dans les couleurs, en plus de l'effet de modulation dynamique.

3. Modulation inversée :

```
voronoi(10, 0.3, 2) // Texture Voronoi
  .modulateHue(osc(5, 0.1, 0), -1.0) // Modulation avec
inversion
  .out()
```

- **Description** : Avec un `amount` négatif, les teintes sont inversées. L'effet devient plus dramatique, et les variations de teinte suivent un décalage opposé au modulateur.

4. Superposition de modulations multiples :

```
osc(30, 0.05, 0.8)
  .modulateHue(gradient(2), 0.3) // Première modulation avec
un dégradé
  .modulateHue(osc(15, 0.2, 0), 0.5, 0.2) // Deuxième
modulation avec une onde
  .out()
```

- **Description** : En combinant plusieurs modulations, on peut obtenir des effets visuels complexes et organiques. Ici, un gradient et une onde influencent ensemble la texture principale.

5. Modulation d'une source vidéo :

```
s0.initCam() // Initialise une caméra comme source
src(s0)
  .modulateHue(osc(10, 0.1, 0), 0.4) // Modulation basée sur
une onde
  .out()
```

- **Description** : L'image capturée par la caméra est modifiée en temps réel, avec des teintes oscillantes dynamiques.

Astuce : Combiner avec d'autres fonctions

Pour enrichir les visuels, vous pouvez utiliser `modulateHue` en combinaison avec :

- **`blend`** : Mélanger la texture modulée avec une autre.

- **`add`** : Ajouter une texture supplémentaire.
- **`layer`** : Superposer plusieurs couches d'effets.

Exemple :

```
osc(10, 0.2, 1).modulateHue(noise(3), 0.8)
  .layer(gradient(1).scale(0.5))
  .out()
```

Effets obtenus grâce à `modulateHue` :

1. **Transitions douces de couleurs** : Idéal pour des effets fluides ou hypnotiques.
2. **Interactions dynamiques** : Les textures interagissent en temps réel pour créer des visuels complexes.
3. **Rythmes chromatiques** : Parfait pour synchroniser avec de la musique ou des entrées extérieures.

Si vous avez un objectif précis pour vos visuels, n'hésitez pas à le mentionner : je peux vous aider à ajuster les paramètres ou proposer d'autres variations.

EXTERNAL SOURCES

INITCAM

Dans **Hydra Video Synth**, la fonction `initCam()` est utilisée pour initialiser et intégrer une caméra en tant que source vidéo dans vos compositions visuelles. Voici une explication révisée, avec des exemples conformes aux fonctionnalités existantes dans Hydra.

Fonctionnement de `initCam()`

1. **Initialisation de la caméra** :
   Lorsque vous appelez `initCam()`, Hydra tente d'accéder à une caméra connectée à votre ordinateur, comme une webcam intégrée ou externe.

2. **Création d'une source** :
   Une fois initialisée, la caméra est associée à une des sources vidéo (par exemple `s0`, `s1`, etc.). Vous pouvez utiliser cette source dans vos shaders pour générer des visuels interactifs ou appliquer des effets.

Exemples de base :

Affichage direct de la caméra :

```
s0.initCam()
src(s0).out()
```

- Ici, la vidéo en direct de la caméra (`s0`) est affichée sans modification.

Application d'effets visuels simples :

Vous pouvez manipuler la source de la caméra pour créer des visuels plus complexes :

```
s0.initCam()
// Ajout d'une rotation et d'une rétroaction (feedback)
src(s0)
  .rotate(() => Math.sin(time) * 0.1) // Rotation dynamique
  .modulate(src(o0), 0.2)             // Rétroaction
  .out()
```

Exemples avancés :

Mélange avec un oscillateur :

Combinez l'entrée caméra avec un oscillateur pour un effet dynamique :

```
s0.initCam()
osc(10, 0.1, 0.8)          // Création d'un oscillateur
  .mult(src(s0))            // Multiplication avec la vidéo
de la caméra
  .modulate(noise(3), 0.3)  // Modulation avec du bruit
  .out()
```

Remarques importantes :

- **Permissions** : Assurez-vous que votre navigateur a les permissions nécessaires pour accéder à la caméra.
- **Limites techniques** : Si plusieurs caméras sont connectées, Hydra utilise par défaut la première caméra détectée.
- **Compatibilité** : Certaines configurations ou navigateurs (comme Firefox) peuvent présenter des restrictions pour accéder à la caméra.

Conclusion :

La fonction `initCam()` est un excellent outil pour intégrer des flux vidéo en direct dans Hydra et les manipuler en temps réel, permettant de créer des performances visuelles interactives et immersives.

INITIMAGE/INITVIDEO

Dans **Hydra Video Synth**, la fonction `initImage()` est utilisée pour charger une image en tant que source visuelle dans vos compositions. Cela permet d'intégrer une image statique comme texture ou élément visuel dans les manipulations graphiques et les effets générés en temps réel.

Fonctionnement de `initImage()et initVideo()`

1. **Chargement d'une image/d'une video** :
   Vous fournissez un chemin d'accès ou une URL vers une image que vous souhaitez utiliser. Hydra charge cette image et l'associe à une des sources vidéo (`s0`, `s1`, etc.).

2. **Utilisation de la source** :
   Une fois l'image chargée, elle peut être utilisée dans vos shaders et manipulée avec les fonctions de transformation, de mélange et d'effets de Hydra.

Exemple simple :

Afficher une image :

```
s0.initImage("path/to/image.jpg") // Remplacez par le chemin
ou l'URL de votre image ou de votre video
src(s0).out()
```

- Cela charge et affiche l'image telle qu'elle est.

ATTENTION AUX DROITS ASSOCIES AUX IMAGES OU VIDEOS, ELLES PEUVENT ETRE SOURCE DE RESTRICTION DANS LEUR IMPORTATION!! PERSONELLEMENT JE PRIVILEGIE: https://commons.wikimedia.org/wiki/Main_Page POUR IMPORTER PHOTOS OU VIDEOS. VOUS POUVEZ EGALEMENT UPLOADER VOS PROPRES IMAGES ET VIDEOS SUR LE SITE EN VUE DE LES UTILISER PAR LA SUITE EN LIGNE.

Exemple avec manipulation :

Ajouter des transformations à l'image :

Vous pouvez appliquer des effets ou modifier l'image chargée pour créer des visuels dynamiques :

```
s0.initImage("https://upload.wikimedia.org/wikipedia/commons/
0/02/1966_Buick_Skylark_convertible_%2814986026094%29.jpg")
// Appliquer une rotation et des effets de modulateur
```

```
src(s0)
  .rotate(() => Math.sin(time) * 0.2) // Rotation dynamique
basée sur le temps
  .scale(1.2)                         // Échelle augmentée
  .modulate(osc(10, 0.1, 0.5))        // Modulation avec un
oscillateur
  .out()
```

Exemple avancé :

Mélanger une image avec d'autres sources :

Combinez l'image avec une caméra ou un oscillateur pour une composition plus complexe :

```
s0.initImage("https://upload.wikimedia.org/wikipedia/commons/
9/99/Plymouth_Special_De_Luxe_4-Door_Touring_Sedan_1936.jpg")
s1.initCam()
// Mélange de l'image avec la vidéo de la caméra
src(s0)
  .blend(src(s1), 0.5)  // Mélange à 50 % avec la caméra
  .modulate(osc(15, 0.1), 0.3) // Modulation avec un
oscillateur
  .out()
```

Utilisations possibles :

1. **Arrière-plan visuel** : Charger une image comme base pour des effets animés.
2. **Textures et masques** : Utiliser une image pour moduler ou texturer d'autres sources.
3. **Mélanges dynamiques** : Combiner des images avec d'autres sources vidéo (caméra, oscillateurs, bruit, etc.).

Remarques importantes :

- **Chemin ou URL** : L'image doit être accessible depuis le chemin ou l'URL spécifié.
- **Compatibilité des formats** : Hydra supporte les formats d'image courants comme JPG et PNG.
- **Taille de l'image** : Les grandes images peuvent être réduites ou déformées selon les transformations appliquées.

Conclusion :

La fonction `initImage()/initVideo` élargit les possibilités de création dans Hydra en intégrant des images statiques comme sources visuelles. Elle est particulièrement utile pour combiner des visuels statiques avec des animations et effets en temps réel, rendant vos performances visuelles encore plus riches et diversifiées.

INITSCREEN

A VERIFIER!!

Dans **Hydra**, la fonction `initScreen()` initialise et configure l'écran ou le canevas sur lequel les visuels générés par Hydra sont affichés. Cette fonction est généralement utilisée au début pour configurer correctement l'environnement de rendu visuel. Voici ses principaux rôles :

1. **Création de la toile de rendu (canvas)** :(?)
   `initScreen()` crée le canevas HTML où Hydra va afficher les visuels. Cela peut inclure la gestion des dimensions du canevas, son placement dans la page, et les paramètres visuels de base.

2. **Initialisation des paramètres de rendu** :
   La fonction configure l'environnement de WebGL nécessaire pour le rendu des graphismes. Cela comprend la gestion des shaders et des textures pour générer les visuels dynamiques.

3. **Connexion au DOM** (?)
   Si vous travaillez dans un navigateur ou une interface intégrée, `initScreen()` associe le canevas à un élément HTML existant (ou en crée un si nécessaire).

Exemple :

Afficher une source et la transformer :

```
//choisir une fenêtre//select a window
s0.initScreen()
src(s0).pixelate(4)
.out()
```

SYNTH SETTINGS

RENDER

Dans **Hydra**, une bibliothèque et environnement de codage pour la création visuelle en temps réel, la fonction `render()` est utilisée pour **afficher la sortie visuelle d'un canal dans un autre canal**, ou pour manipuler des contenus visuels en combinant des flux vidéo ou des textures générés par d'autres parties de la scène.

Fonctionnement de `render()`

1. **Source d'entrée** :
   Par défaut, Hydra dispose de quatre canaux principaux (ou buffers) : `o0`, `o1`, `o2`, et `o3`. Ces canaux contiennent des flux visuels générés par les chaînes de commandes.

2. **Manipulations avancées** :
   En combinant `render()` avec d'autres fonctions comme `src()`, `blend()`, ou `modulate()`, vous pouvez superposer des couches, créer des effets de feedback visuel, ou mélanger différents éléments pour enrichir votre visuel.

Exemples pratiques

1. Utilisation de base :

```
osc(10, 0.1, 1.5).out() // Génère une sortie sur o0
render(o0)              // Redirige la sortie vers le canal 1
```

2. Combiner des flux avec `render` et `src` :

```
osc(20, 0.05, 0.8)
     .out(o0)
noise(10)
     .out(o1)
src(o1)
     .blend(src(o0), 0.726)
     .out(o2)
src(o2)
     .mult(src(o2), 0.726)
.pixelate()
     .out(o3)
render() //affiche les 4 fenêtres
```

3. Effet de rétroaction :

```
 osc(30, 0.1, 1)
  .modulate(src(o0), 0.1) // Ajoute un effet de modulation
basé sur o0
  .out()

//render() pour afficher sur la totalité de l'écran
```

Pourquoi utiliser `render()` ?

- **Flexibilité** : Cela permet de travailler avec plusieurs couches visuelles sans se limiter à une seule sortie.
- **Compositions dynamiques** : Créez des effets complexes ou des interactions entre différents canaux.
- **Optimisation des performances** : En rendant un flux dans un autre canal, vous évitez de recalculer certaines parties.

Si vous avez un projet spécifique ou un effet en tête, je peux vous aider à l'implémenter en détaillant l'utilisation de `render()` dans votre contexte !

SETRESOLUTION

Doesn't work on my Mac in chrome ou Safari!

Dans **Hydra**, une plateforme de codage pour la création visuelle en temps réel, la fonction **`setResolution`** permet de définir la résolution de la sortie visuelle (le canvas ou le rendu). Cela contrôle la largeur et la hauteur du rendu en pixels. Voici une explication détaillée :

Utilité de `setResolution`

1. **Contrôle de la qualité de l'image :**
    - Une résolution plus élevée produit des visuels plus nets et détaillés.
    - Une résolution plus basse peut donner un effet pixellisé ou stylisé, ce qui peut être utilisé intentionnellement dans des créations artistiques.
2. **Impact sur les performances :**
    - Une résolution plus basse réduit la charge sur le processeur graphique (GPU), ce qui est utile si votre système a des ressources limitées.
    - Une résolution plus élevée peut nécessiter davantage de puissance de calcul, ce qui pourrait ralentir le rendu.
3. **Adaptation à l'écran ou au contexte de présentation :**
    - Vous pouvez ajuster la résolution pour correspondre aux dimensions d'un écran spécifique ou à une configuration particulière (comme des projecteurs ou des écrans LED).

Syntaxe de `setResolution`

```
setResolution(width, height)
```

- **`width`** : La largeur en pixels.
- **`height`** : La hauteur en pixels.

Par exemple :

```
setResolution(1920, 1080) // Définit la résolution à
1920x1080 (Full HD).
setResolution(800, 600)   // Définit la résolution à 800x600.
```

Comportement par défaut

- Par défaut, Hydra utilise la résolution de votre écran ou fenêtre de rendu.

- Si vous ne définissez pas de résolution avec `setResolution`, Hydra ajuste automatiquement le rendu à la taille de la fenêtre.

Exemples pratiques

1. **Effet pixellisé intentionnel :**

setResolution(320, 240) // Résolution basse pour un rendu pixellisé. osc(10, 0.1, 0.8).out()

```
2. **Création de visuels haute résolution pour un projet :**
```javascript

setResolution(3840, 2160) // Résolution 4K pour des visuels
très détaillés.
voronoi(10).out()
```

Conseils

- Expérimentez avec différentes résolutions pour voir comment cela affecte l'esthétique et les performances.
- Assurez-vous d'adapter la résolution à votre matériel pour éviter des ralentissements inutiles.

Avez-vous besoin d'aide pour un projet spécifique avec Hydra ? 😊

HUSH

Dans **Hydra**, une plateforme de live coding visuel, la commande `hush()` est utilisée pour arrêter tout rendu ou sortie visuelle en cours. C'est une commande pratique pour "nettoyer" l'écran ou interrompre les effets visuels sans avoir à redémarrer le programme ou supprimer manuellement les lignes de code.

Fonctionnalité principale de `hush()` :

- **Stopper le rendu :** Dès que la commande `hush()` est appelée, Hydra arrête immédiatement tous les flux de rendu visuel actifs. Cela inclut les oscillateurs, les shaders, ou tout autre effet visuel généré.
- **Contexte de nettoyage :** Cela peut être utile lorsque vous voulez repartir sur une base propre pour tester un nouveau code ou supprimer des visuels sans quitter votre session.

Exemple d'utilisation :

```
osc(10, 0.1, 0.8).out()
// Après avoir vu le visuel en action, vous voulez
l'arrêter :
hush()
```
```

Cas d'usage pratique :

1. **Interruption rapide :** Pendant une performance live, vous pourriez vouloir arrêter les visuels temporaires sans perdre de temps à éditer le code.
2. **Expérimentation :** Lors de l'écriture et du test de nouveaux visuels, vous pouvez utiliser `hush()` pour arrêter ce qui est affiché avant de lancer un autre visuel.

Limitation :

`hush()` ne supprime pas le code ou les paramètres définis dans votre session. Les définitions de variables, fonctions ou configurations restent actives. Pour un redémarrage complet, vous devrez réinitialiser votre environnement manuellement.

SETFUNCTION

Dans **Hydra**, la commande `setFunction()` permet de définir ou redéfinir une fonction personnalisée pour l'environnement de live coding. C'est une manière de modifier ou d'étendre les fonctionnalités existantes d'Hydra en ajoutant des comportements ou des effets personnalisés, directement accessibles dans votre session.

CF HERE: https://hydra.ojack.xyz/docs/docs/learning/extending-hydra/glsl/

Fonctionnalité principale de `setFunction()` :

- **Ajouter des fonctions personnalisées :** Vous pouvez définir vos propres fonctions et les utiliser comme n'importe quelle autre commande native de Hydra.
- **Redéfinir des fonctions existantes :** Vous pouvez remplacer les comportements par défaut des fonctions intégrées si vous souhaitez personnaliser leurs résultats ou effets.

Syntaxe générale :

```
setFunction(name, func)
```

- **`name`** : Le nom que vous voulez donner à la nouvelle fonction ou à celle que vous souhaitez redéfinir.
- **`func`** : Une fonction JavaScript qui décrit le comportement souhaité.

Exemple d'utilisation venant du document référence d'Hydra:

```
// from https://www.shadertoy.com/view/XsfGzn

setFunction({name: 'chroma',

  type: 'color',

  inputs: [

    ],
```

```
  glsl: `
   float maxrb = max( _c0.r, _c0.b );
   float k = clamp( (_c0.g-maxrb)*5.0, 0.0, 1.0 );
   float dg = _c0.g;
   _c0.g = min( _c0.g, maxrb*0.8 );
   _c0 += vec4(dg - _c0.g);
   return vec4(_c0.rgb, 1.0 - k);
`})
osc(60,0.1,1.5).chroma().out(o0)
```

Cas d'usage pratique :

1. **Réutilisation de code :** Si vous utilisez fréquemment un ensemble d'effets, vous pouvez les regrouper dans une fonction personnalisée pour simplifier votre session.
2. **Redéfinition contextuelle :** Pendant une performance live, vous pouvez ajuster ou redéfinir des fonctions pour répondre à une ambiance ou un style spécifique sans modifier tout votre code.
3. **Extension des fonctionnalités :** Créez des effets ou comportements non natifs à Hydra en tirant parti de JavaScript.

Limitation :

- Les modifications effectuées avec `setFunction()` sont limitées à la session en cours. Si vous redémarrez Hydra, les définitions seront perdues, sauf si elles sont sauvegardées dans un fichier ou un script.

SPEED

Dans **Hydra**, `speed()` est une fonction qui contrôle la **vitesse globale de l'animation** dans le système. Elle agit comme un multiplicateur de temps, influençant toutes les sources, oscillateurs, et autres modulateurs temporels dans votre composition visuelle.

Utilisation :

```
speed = x
```

- **facteur** : Un nombre (positif, négatif ou 0) qui détermine la vitesse relative de l'animation.

Effets du paramètre `facteur` :

1. **Vitesse normale** :
    - Par défaut, la vitesse est réglée à `1`.
    - Si vous ne spécifiez rien, les animations avancent à leur cadence normale.
2. **Accélérer** :
    - Une valeur supérieure à `1` accélère l'animation.
    - Exemple : `speed = 2 // Double la vitesse.`
    -
3. **Ralentir** :
    - Une valeur entre `0` et `1` ralentit l'animation.
    - Exemple : `speed = 0.5 // Réduit la vitesse de moitié.`
    -
4. **Mettre en pause** :
    - Si vous réglez la vitesse sur `0`, toutes les animations se figent.
    - Exemple : `speed = 0`
    -
5. **Inverser l'animation** :
    - Une valeur négative fait jouer les animations en sens inverse.
    - Exemple : `speed(-1) // L'animation recule à la vitesse normale.`
    -

Exemple pratique :

Imaginons que vous avez une animation définie comme suit :

```
osc(10, 0.1, 1).out()
```

- Pour accélérer la vitesse globale :
  `speed = 2`
-
- Pour inverser et ralentir l'animation :
  `speed = -0.5`
-

Points importants :

- `speed()` affecte **tout le projet**, pas seulement une source ou une chaîne en particulier.
- Si vous voulez changer la vitesse d'une animation spécifique, utilisez des paramètres de modulateurs locaux (comme dans `osc(freq, speed, amp)`).

En résumé, `speed()` est un outil puissant pour contrôler la dynamique globale de vos visuels, idéal pour ajuster l'ambiance ou expérimenter avec le rythme temporel dans Hydra.

BPM

https://hydra.ojack.xyz/functions/#functions/bpm/0

The speed of all arrays in a sketch can be changed using the `bpm` parameter of hydra synth.

bpm = 60
osc(60,0.1,[0,1.5]).out(o0)

WIDTH/HEIGHT

Examples tirés de la documentation d'Hydra. ChatGPT nous a donné une réponse totalement erronée!

Scroller dans le sens de la largeur:

```
shape(99).scrollX(() => -mouse.x / width).out(o0)
```

Scroller dans le sens de la hauteur:

```
shape(99).scrollY(() => -mouse.y / height).out(o0)
```

TIME

Rôle de `time` dans Hydra

- **`time`** est une variable dynamique qui représente le temps écoulé (en secondes) depuis le lancement du programme ou du rendu graphique.
- Elle est mise à jour en continu et peut être utilisée pour introduire des variations ou des animations dans les visuels.

Explication de l'exemple

Voici une décomposition de la ligne de code :

```
shape(2, 0.8)
  .kaleid(() => 6 + Math.sin(time) * 4)
  .out(o0)
```

1. **`shape(2, 0.8)`** :
    - Crée une forme géométrique, ici un **polygone** avec 2 côtés (donc une ligne droite).
    - Le second paramètre `0.8` contrôle la "netteté" des bords de la forme.
2. **`.kaleid(() => 6 + Math.sin(time) * 4)`** :

- Applique un effet de **kaléidoscope** à la forme.
- La fonction passée à `kaleid` détermine dynamiquement le nombre de "segments" (symétries) dans le kaléidoscope.
- **`Math.sin(time)`** oscille entre -1 et 1 au fil du temps, créant un mouvement fluide et cyclique.
- **`6 + Math.sin(time) * 4`** ajuste dynamiquement le nombre de segments, oscillant entre 2 (6 - 4) et 10 (6 + 4).

3. **`.out(o0)`** :
   - Envoie le rendu final au canal de sortie `o0`, qui affiche l'animation à l'écran.

Résultat visuel attendu

- Une forme (ici une ligne droite) se transforme en un motif kaléidoscopique dont le nombre de segments change de manière fluide et continue au rythme de la fonction sinusoïdale.

Pourquoi utiliser `time` ?

- La variable `time` permet de créer une animation dynamique et organique qui évolue sans intervention manuelle.
- Sans `time`, le nombre de segments du kaléidoscope resterait fixe, et l'animation manquerait de fluidité.

Si vous avez besoin de détails supplémentaires ou d'autres exemples, n'hésitez pas à demander ! 😊

MOUSE

Tiré de la documentation d'hydra:
https://hydra.ojack.xyz/functions/#functions/mouse/0

Exemple:

```
shape(99).scroll(
  () => -mouse.x / width,
  () => -mouse.y / height)
  .out(o0)
```

ARRAY

Fast

Exemple:

```
shape([3, 6, 9].fast(2)).out()
```

Cela s'explique par la structure de Hydra où **`fast()` agit directement sur un tableau** (**`[3, 6, 9]`**), et non comme une méthode indépendante de `shape()`. Voici une explication plus détaillée :

Décryptage du code :

```
shape([3, 6, 9].fast(2)).out()
```

1. **`[3, 6, 9]` :**
   - Un tableau contenant les valeurs qui seront utilisées pour les paramètres de la fonction `shape()`.
   - Ici, les valeurs correspondent au nombre de côtés des formes : un triangle (3), un hexagone (6), et un nonagone (9).
2. **`.fast(2)` :**
   - Appliqué au tableau `[3, 6, 9]`, cette méthode fait en sorte que Hydra passe d'une valeur à l'autre dans le tableau à **deux fois la vitesse normale**.
3. **`shape()` :**
   - Utilise le tableau en alternant dynamiquement entre les valeurs, créant des formes avec des nombres de côtés différents.
4. **`out()` :**
   - Affiche le résultat de cette composition sur l'écran.

Ce qui apparaît visuellement

- Une alternance rapide entre un triangle, un hexagone et un nonagone.
- Si vous augmentez ou diminuez la valeur de `.fast()`, cela changera la vitesse à laquelle ces formes s'alternent.

Exemple avancé : combinaison avec d'autres transformations

Vous pouvez enrichir cet effet en ajoutant des rotations ou des modulations :

```
shape([3, 6, 9].fast(4))
  .rotate(0.1)
  .modulate(osc(5, 0.1))
  .out()
```

Explication :

1. **`[3, 6, 9].fast(4)` :**
   - La transition entre les formes est encore plus rapide grâce à `.fast(4)`.
2. **`rotate(0.1)` :**
   - Ajoute une rotation constante à la forme.
3. **`modulate(osc(5, 0.1))` :**

- L'oscillateur apporte une modulation dynamique à la composition.

SMOOTH

Dans **Hydra**, la fonction `smooth()` est utilisée pour **adoucir les transitions** entre des valeurs, créant des effets visuels plus fluides et progressifs. Elle est particulièrement utile lorsqu'on travaille avec des modulations ou des tableaux pour éviter des sauts brusques ou des changements abrupts dans une animation.

Fonctionnement de `smooth()`

- **Adoucissement temporel :** La méthode `smooth()` agit comme un filtre qui applique une interpolation linéaire (ou similaire) entre les valeurs, ralentissant les transitions pour les rendre plus fluides.
- **Application :**
  - Elle est souvent utilisée en conjonction avec des tableaux (par exemple, `[value1, value2].smooth()`).
  - Peut être combinée avec des oscillateurs ou des sources visuelles pour produire des animations plus douces.

Exemple de base avec un tableau

```
shape([3, 6, 9].smooth(0.5)).out()
```

Explication :

1. **`[3, 6, 9]` :**
   - Un tableau contenant des valeurs qui modifient dynamiquement le nombre de côtés de la forme.
2. **`.smooth(0.5)` :**
   - Applique un adoucissement aux transitions entre les valeurs du tableau. La valeur `0.5` contrôle le degré d'adoucissement :
     - Une valeur plus faible (par exemple, `0.1`) rend les transitions plus rapides.
     - Une valeur plus élevée (par exemple, `1`) les rend plus lentes et progressives.
3. **`shape()` :**
   - Utilise ces valeurs pour déterminer le nombre de côtés de la forme (triangle, hexagone, nonagone), mais les transitions entre elles sont maintenant douces.
4. **`out()` :**
   - Affiche le résultat à l'écran.

Exemple combiné avec des oscillateurs

```
osc(10, 0.1, 1)
  .modulate(shape([3, 6, 9].smooth(0.3)))
  .out()
```

Explication :

- `[3, 6, 9].smooth(0.3)` :
  - Adoucit les transitions entre les différentes formes pour éviter des changements brusques dans la modulation.
- **Résultat visuel :**
  - Les oscillations de `osc()` sont modulées par des formes qui évoluent de manière fluide, créant un effet visuel organique.

Points importants sur `smooth()` :

1. **Adoucissement contrôlé :**
   - La valeur passée à `smooth()` contrôle l'intensité de l'effet. Une valeur trop élevée peut ralentir les transitions au point de les rendre imperceptibles.
2. **Applications créatives :**
   - Améliore l'esthétique visuelle en supprimant les transitions saccadées.
   - Idéal pour des animations synchronisées à la musique ou pour des effets méditatifs et immersifs.
3. **Combinaisons :**
   - Combinez `smooth()` avec des modulateurs, des oscillateurs ou des transformations pour des animations complexes et fluides.

EASE

Dans Hydra, l'opérateur `ease` n'est pas utilisé de manière isolée, mais fait effectivement partie d'un tableau dans des contextes spécifiques. Cela est dû au fait qu'il est généralement employé en association avec d'autres paramètres dynamiques pour moduler des transitions ou interpolations complexes.

Voici quelques explications et exemples pour clarifier l'utilisation de `ease` comme une partie d'un tableau :

Fonctionnement avec des tableaux

Hydra permet l'utilisation de **tableaux de valeurs dynamiques** pour créer des comportements modulés. Lorsqu'`ease` est utilisé dans un tableau, il agit sur un ensemble de valeurs et applique des transitions interpolées pour chacune d'entre elles.

Exemple d'utilisation dans un tableau

1. **Interpoler plusieurs valeurs d'une couleur** :
2.
```
osc(10, 0.1, 1)
```
3.
```
  .colorama([0.1, 0.5, 0.9].ease('easeInOutCubic'))
```
4.
```
  .out()
```

    - Ici, un tableau `[0.1, 0.5, 0.9]` est interpolé grâce à `ease`, ce qui rend les transitions entre les valeurs de `colorama` plus fluides au fil du temps. Et 'easeInOutCubic' fait référence à la fonction mathématique opérant les interpolations entre les valeurs. Pour une liste détaillée de ces fonctions « d'assouplissement »
    - spécifiant le taux de variation d'un paramètre au fil du temps cf: https://easings.net/

5. **Animation complexe avec un tableau dynamique** :
6.
```
shape(4)
```
7.
```
  .scale([0.5, 1, 1.5].ease('easeInOutQuint'))
```
8.
```
  .out()
```

    - Dans cet exemple, la taille du `shape` passe progressivement entre les valeurs 0.5, 1, et 1.5 grâce à `ease`, créant une animation cyclique.

Notes importantes

- **Syntaxe correcte** : Assurez-vous d'utiliser `.ease()` avec un tableau pour bénéficier de ses propriétés d'interpolation.
- **Effets visuels dynamiques** : En combinant `ease` avec d'autres paramètres comme `modulate`, `color`, ou `scale`, vous pouvez produire des visuels très complexes et organiques.

OFFSET

From the docs: https://hydra.ojack.xyz/docs/docs/learning/sequencing-and-interactivity/arrays/

Another one of the methods Hydra adds to Arrays, allows you to offset the timing at which Hydra will switch from one element of the Array to the next one. The method .offset takes a Number from 0 to 1.

Une autre des méthodes que Hydra ajoute aux tableaux vous permet de décaler le moment auquel Hydra passera d'un élément du tableau au suivant. La méthode .offset prend un nombre compris entre 0 et 1.

Exemple:

```
noise([2, 10, 30].offset([1, 0.25, 0.5]))

.color(1,0,3)

  .out()
```

Chaque valeur du tableau crée un décalage unique sur le motif dans le temps et dans l'espace comme dans l'exemple ci-dessous:

```
shape(4)
  .scale(0.5)
  .repeat([2, 3, 4].offset([0.2, 0.4, 0.6]))
  .out()
```

FIT

Vous avez tout à fait raison, et je m'excuse pour l'erreur précédente. Dans **Hydra Video Synth**, la méthode `.fit()` appliquée à un **Array** est utilisée pour **redimensionner les valeurs d'un tableau afin de les ajuster dans une plage spécifiée**. Cela permet de manipuler des données pour les rendre utilisables dans des fonctions Hydra.

Fonctionnement de `.fit()` :

La méthode `.fit()` prend deux arguments, **min** et **max**, et mappe les valeurs du tableau existant à cette plage.

Cela est utile, par exemple, lorsque vous travaillez avec des données générées aléatoirement ou avec des valeurs qui ne correspondent pas à l'échelle nécessaire pour un effet spécifique.

Syntaxe générale :

```
array.fit(min, max)
```

Exemple d'utilisation :

Supposons que vous ayez un tableau avec des valeurs allant de 0 à 10, mais que vous souhaitez les adapter à une plage allant de 0 à 1 (plus utile dans Hydra).

```
[0, 2, 5, 10].fit(0, 1)
// Résultat : [0, 0.2, 0.5, 1]
```

Cela convertit les valeurs initiales dans une plage proportionnelle à la nouvelle plage spécifiée.

Exemple pratique dans Hydra inspiré de:

https://hydra.ojack.xyz/?code=YnBtJTIwJTNEJTIwMTIwJTBBYXJyJTIwJTNEJTIwKCklM0QlM0UlMjAlNUIxJTJDMiUyQzQlMkM4JTJDMTYlMkMzMiUyQzY0JTJDMTI4JTJDMjU2JTJDNTEyJTVEJTBBb3NjKDUwJTJDLjElMkNhcnIoKS5maXQoMCUyQ01hdGguUEkpKSUwQSUwOS5zY2FsZShhcnIoKS5maXQoMSUyQzIpKSUwQSUwOS5vdXQoKQ==

```
bpm = 120
arr = ()=> [1,2,4,8,16,32,64,128,256,512]
osc(50,.1,arr().fit(0,100))
     .scale(arr().fit(1,10))
     .out()
```

Pourquoi c'est utile :

- **Adaptation des données** : Permet de rendre les données exploitables pour des fonctions Hydra comme `modulate`, `osc`, ou `scale`.
- **Créativité** : Donne plus de contrôle sur l'échelle et la portée des effets visuels générés dynamiquement.

FFT/SETSMOOTH/SETCUTOFF/SETBINS/SETSCALE

https://hydra.ojack.xyz/docs/docs/learning/sequencing-and-interactivity/audio/

Réactivité audio #
La fonctionnalité FFT est disponible via un objet audio accessible via « a ». L'éditeur utilise https://github.com/meyda/meyda pour l'analyse audio. Pour afficher les bins fft,

```
a.show()
```

Définissez le nombre de bins fft :

```
a.setBins(6)
```

Accédez à la valeur du bin le plus à gauche (fréquence la plus basse) :

```
a.fft[0]
```

Utilisez la valeur pour contrôler une variable :

```
osc(10, 0, () => a.fft[0]*4)
.out()
```

Il est possible de calibrer la réactivité en modifiant la valeur minimale et maximale détectée. (Représentée par des lignes floues sur le fft). Pour définir la valeur minimale détectée :

```
a.setCutoff(4)
```

Le réglage de l'échelle modifie la plage détectée.

```
a.setScale(2)
```

Le fft[] renverra une valeur comprise entre 0 et 1, où 0 représente la coupure et 1 correspond au maximum.

Vous pouvez définir un lissage entre les lectures de niveau audio (valeurs comprises entre 0 et 1). 0 correspond à aucun lissage (plus instable, temps de réaction plus rapide), tandis que 1 signifie que la valeur ne changera jamais.

```
a.setSmooth(0.8)
```

Pour masquer la forme d'onde audio :

```
a.hide()
a.setBins(5)  // nombre de bacs (bandes) pour séparer le spectre audio

noise(2)
.modulate(o0,()=>a.fft[1]*.5)  // écoute de la 2e bande
.out()

a.setSmooth(.8)  // lissage de la réactivité audio de 0 à 1, utilise une interpolation linéaire
a.setScale(8)  // limite supérieure du volume sonore (correspond à 0)
a.setCutoff(0.1)  // volume sonore à partir duquel commencer à écouter (correspond à 0)
a.show() // afficher ce qu'écoute Hydra
// a.hide()

render(o0)
```