50-Texturation procédurale et modélisation

Dans ce didacticiel, nous allons examiner différentes opérations pouvant être utilisées pour construire un modèle procédural d'une texture ou d'une certaine forme de données géométriques.

Les techniques procédurales sont un moyen puissant de définir certains aspects d'un modèle généré par ordinateur au moyen d'algorithmes et/ou de fonctions mathématiques. Contrairement à l'utilisation de données préexistantes telles que des images statiques ou des photographies, les modèles procéduraux peuvent générer une complexité visuelle de résolution arbitraire et de variation infinie. En conjonction avec des contrôles paramétriques, de tels modèles peuvent être utilisés pour construire une interface flexible permettant de contrôler des comportements complexes et de capturer un effet spécial.

Jitter fournit un ensemble complet de fonctions de base et de générateurs qui sont exposés via l'objet *jit.bfg*. Chaque fonction effectue une opération ponctuelle dans un espace à n dimensions dont l'évaluation est indépendante des résultats voisins. Cela signifie que ces opérations peuvent être effectuées sur un nombre quelconque de dimensions, sur n'importe quelle coordonnée, sans qu'il soit besoin de faire référence à les calculs existants. De plus, puisqu'ils partagent tous une interface commune, ces objets peuvent être combinés ensemble et évalués dans un graphe de fonctions en faisant référence à plusieurs objets *jit.bfg*.

Il existe plusieurs catégories de fonctions, chacune d'entre elles caractérisée par une utilisation différente. Ces catégories comprennent les opérations ***fractales***, de ***bruit***, de ***filtre***, de ***transfert*** et de ***distance***. Les fonctions contenues dans ces dossiers peuvent être transmises par leur nom à *jit.bfg*, soit entièrement qualifiées (category.classname), soit détendues (classname).

Avant d'examiner ces catégories en détail, nous allons d'abord explorer l'interface générale de *jit.bfg* et montrer comment créer différents types de fonctions procédurales et remplir une *jit.matrix* avec nos résultats.

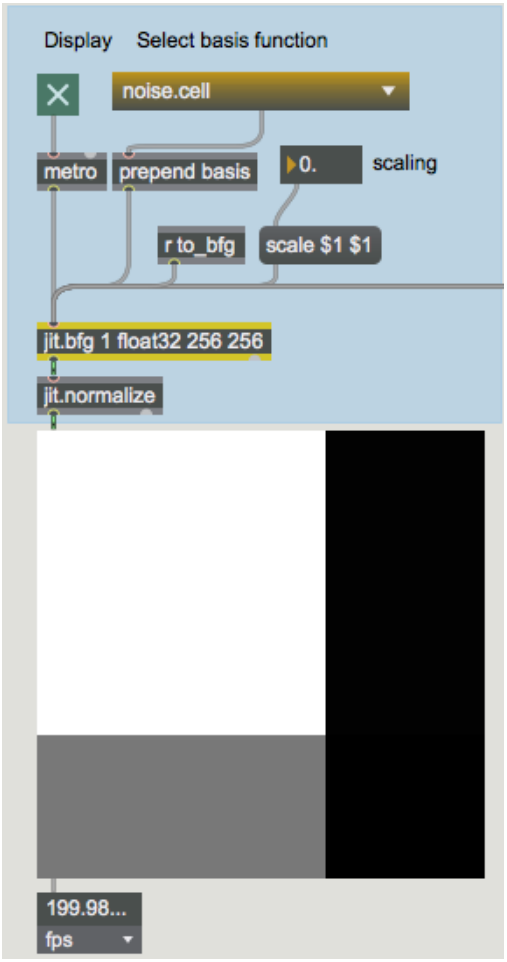
jit.bfg

Une fois le patch chargé, regardez les différents objets utilisés. Remarquez que nous avons un objet *metro* attaché à l'objet *jit.bfg*. Une fois activé, un message **bang** notifiera à *jit.bfg* d'évaluer et de sortir une matrice Jitter comme la plupart des autres objets Jitter. Dans cet exemple, *jit.bfg* a été configuré pour générer une matrice à plan unique de type **float32** et de taille **128x128**.

• Cliquez sur le *toggle* connecté à l'objet *metro* pour commencer à envoyer des messages **bang** à *jit.bfg*.

Notez que l'objet *jit.pwindow* reste de couleur noire unie! Puisq'on n'a pas dit à *jit.bfg* quelle fonction de base évaluer, *jit.bfg* ne génère pas de matrice et *jit.pwindow* reste inchangé.

• Sélectionnez la fonction **noise.cell basis** dans la liste de l'objet *umenu*.



Une fonction de base évaluée.

Maintenant que *jit.bfg* a reçu une fonction à évaluer, nous pouvons voir les résultats de son calcul dans *jit.pwindow*. En interne, *jit.bfg* génère une série de coordonnées cartésiennes qu'il passe à la fonction de base **basis** spécifiée pendant de son évaluation.

Si nous le voulions, nous pourrions ajuster ces coordonnées et demander à *jit.bfg* d'effectuer l'évaluation sur un domaine différent.

• Modifiez la valeur de la boîte de *nombre* connectée à la boîte de *message* **scale**.

Remarquez que les résultats de *jit.bfg* changent lorsque nous ajustons le domaine.

• Sélectionnez la fonction **distance.euclidean basis** dans la liste de l'objet *umenu*.
• Modifiez à nouveau la valeur de la boîte de *nombre* connectée à la boîte de *message* **scale**.

Remarquez que les valeurs positives de la boîte de *nombre* **scale** ont peu d'effet sur les résultats affichés dans *jit.pwindow*, tandis que les valeurs négatives font passer les composants de l'image du blanc au noir. Que se passe-t-il? La distance devrait toujours être positive et augmenter vers l'extérieur depuis l'origine, n'est-ce-pas?

jit.normalize

La sortie de *jit.bfg* va dans un objet *jit.normalize* connecté à jit.pwindow. Cet objet va examiner une matrice entrante et mettre à l'échelle les valeurs minimales et maximales dans une plage normalisée de **0 -1**.

Lorsque nous avons modifié les valeurs **scale** envoyées à *jit.bfg* pour évaluer la fonction **distance.euclidean** de positive à négative, les valeurs les plus élevées et les plus basses qui étaient émises par *jit.bfg* ont changé lorsque nous avons traverser le point d'origine. Puisque *jit.normalize* met toujours à l'échelle le maximum de la matrice d'entrée à **1** et minimum à **0**, nos couleurs se sont inversées.

Puisque la plage de sortie de *jit.bfg* peut donner des résultats extrêmement importants, en particulier lors de l'évaluation de fonctions non limitées telles que les fractales, nous devons normaliser notre sortie afin d'afficher les résultats.

Catégories de base

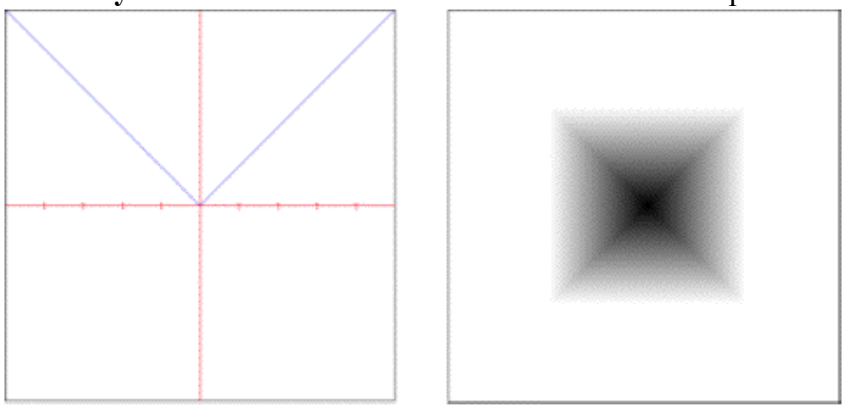
Maintenant que nous sommes familiarisés avec l'interface de base pour configurer *jit.bfg* et spécifier une fonction de base à évaluer, examinons le contenu de chaque catégorie de fonction.

Fonctions de distance

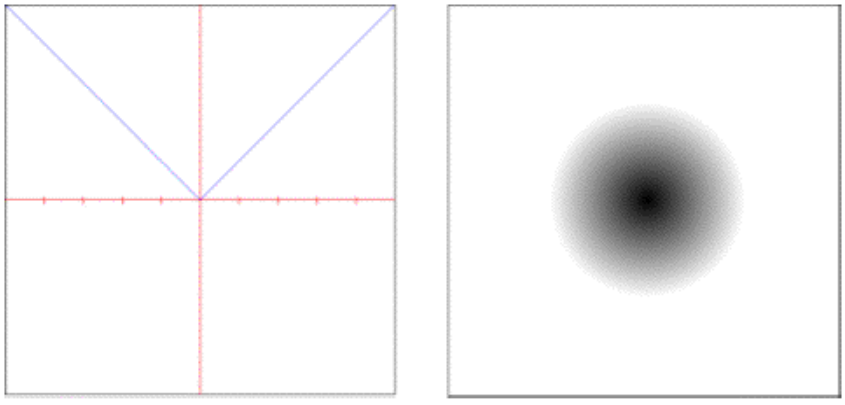
Les fonctions de la catégorie **distance** définissent chacune une métrique unique pour déterminer la différence de position entre un point donné et l'origine globale.

Les descriptions de chacune de ces fonctions sont fournies dans la liste suivante.

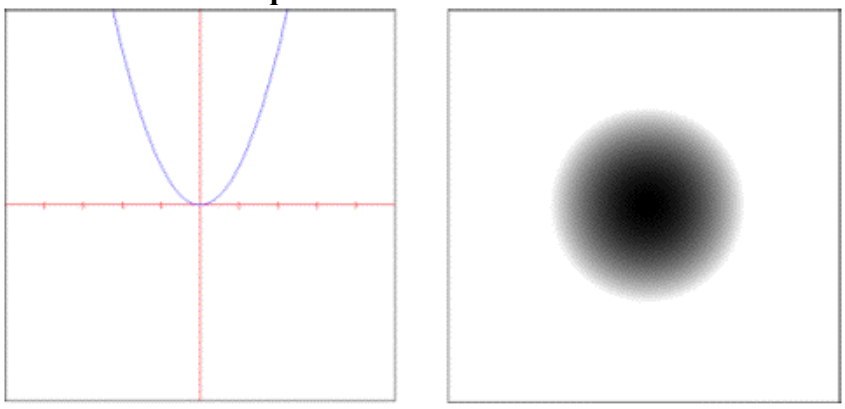
• **chebychev**: différence maximale absolue entre deux points.



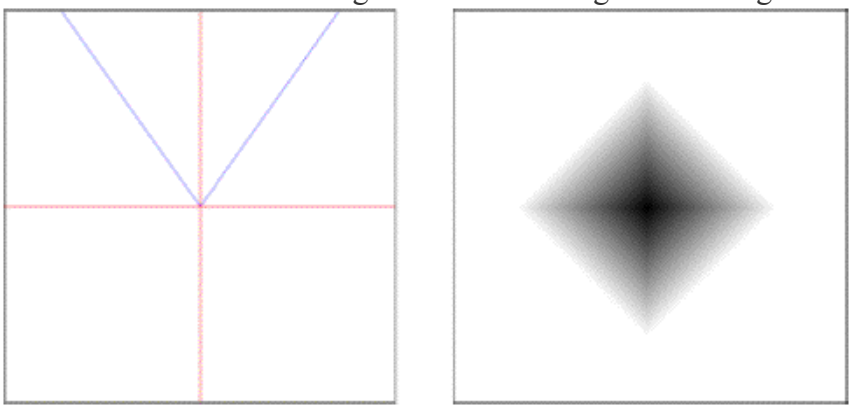
• **euclidean**: véritable distance en ligne droite dans l'espace euclidien.



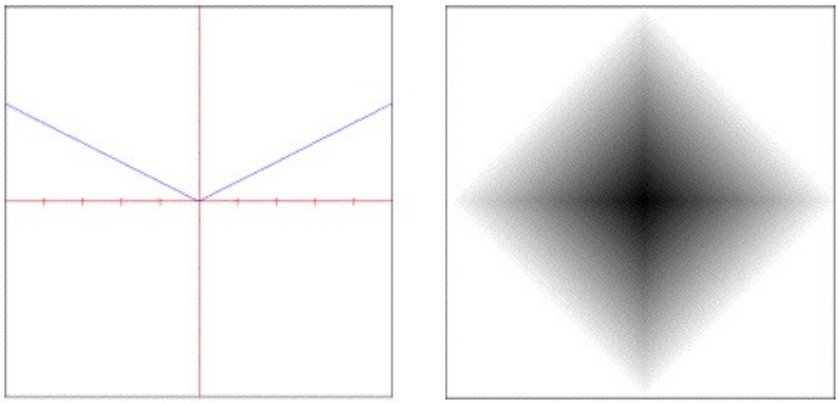
• **euclidean.squared**: distance euclidienne au carré.



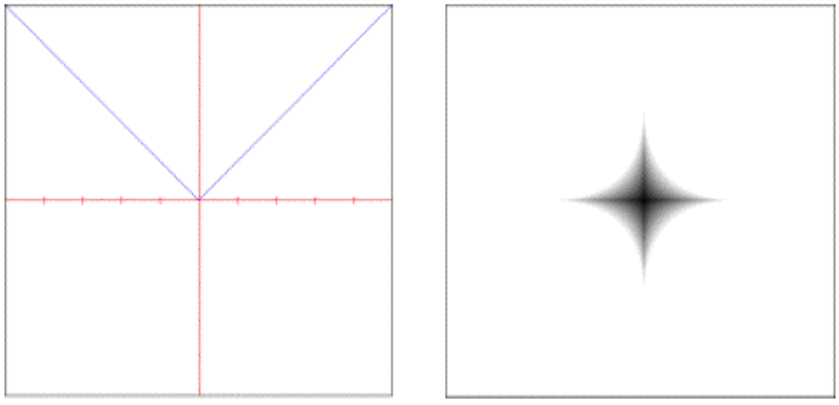
• **manhattan**: distance rectiligne mesurée le long d'axes à angle droit.



• **manhattan.radial**: distance de manhattan avec contrôle de la chute du rayon.



• **minkovsky**: distance contrôlée de manière exponentielle.



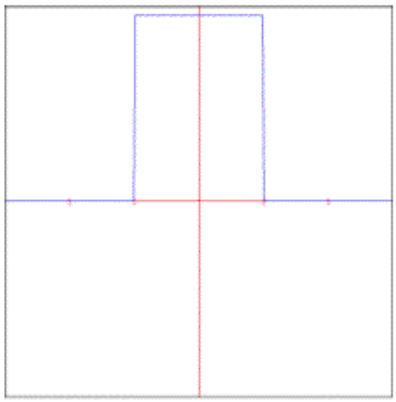
L'objet **noise.voronoi** nécessite que l'un des objets distance soit spécifié dans le cadre de son évaluation.

Fonctions de filtrage

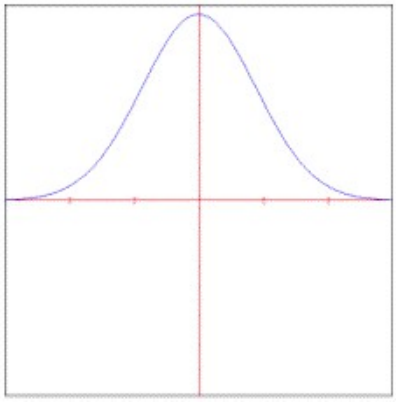
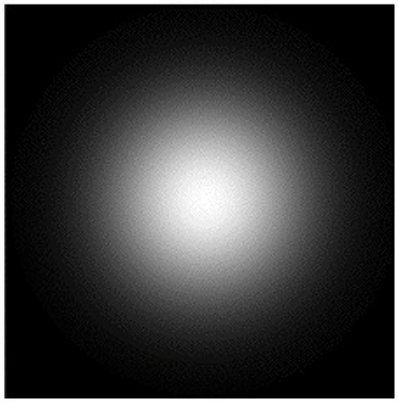
La catégorie **filter** contient des filtres de traitement du signal qui peuvent être utilisés pour effectuer l'échantillonnage et la reconstruction d'images ou pour créer des noyaux pré-calculés pour une convolution générale.

Des descriptions de chacune de ces fonctions sont fournies dans la liste suivante.

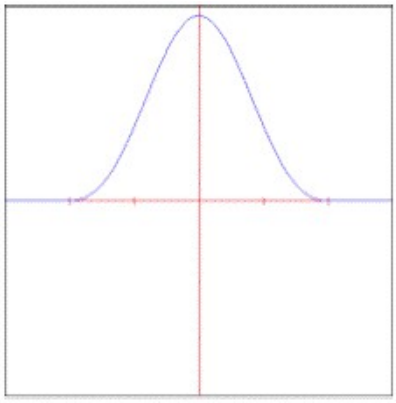
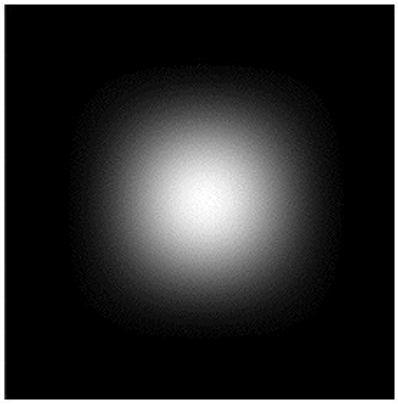
• **box**: additionne tous les échantillons dans la zone du filtre avec un poids égal.



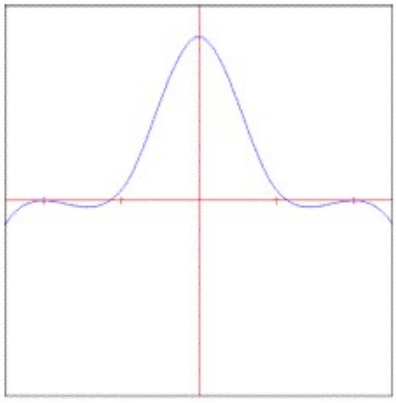
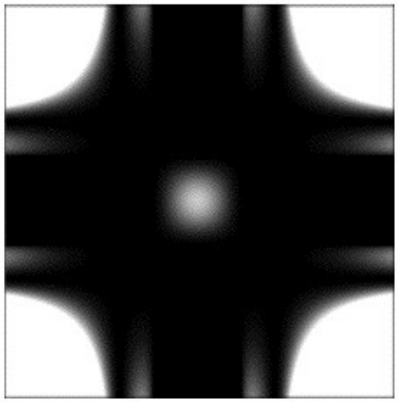

• **gaussian**: pondère les échantillons dans la zone de filtrage en utilisant une courbe en cloche.

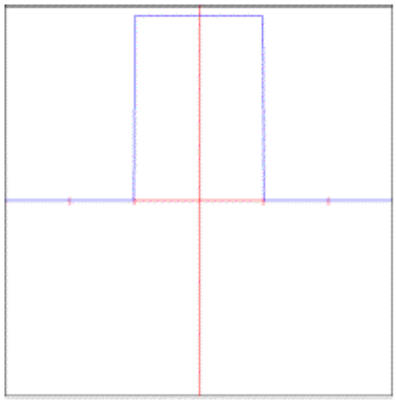
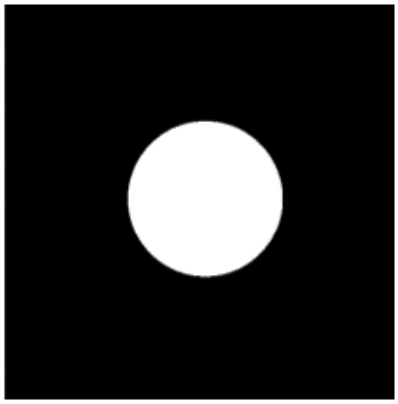
• **lanczossinc**: pondère les échantillons en utilisant une courbe sinc à fenêtre étroite.

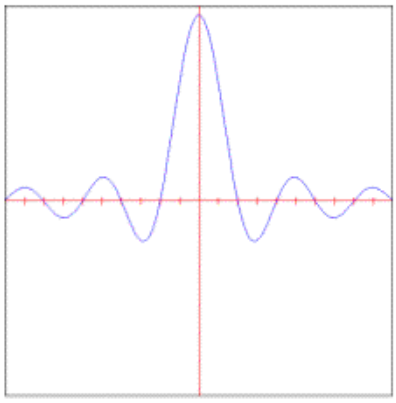
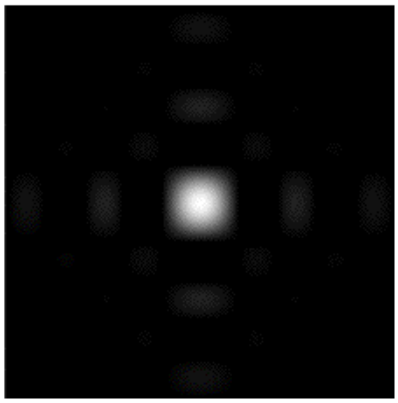
• **mitchell**: pondère les échantillons en utilisant un polynôme cubique contrôlable.

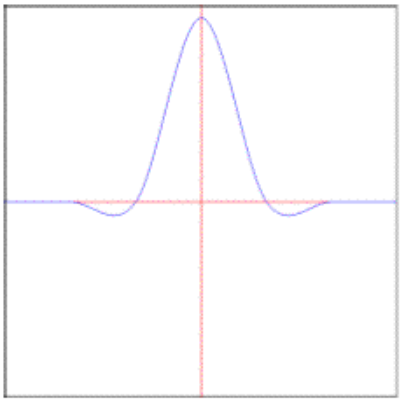
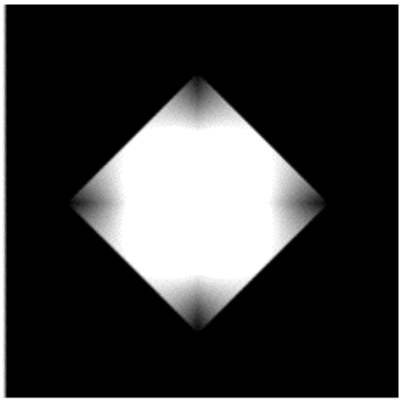
• **disk**: additionne tous les échantillons à l'intérieur du rayon du filtre avec un poids égal.

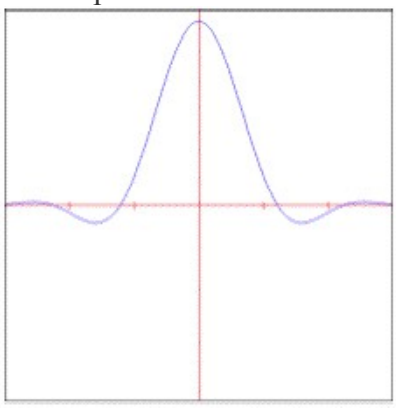
• **sinc**: pondère les échantillons à l'aide d'une courbe sinc non fenêtrée.

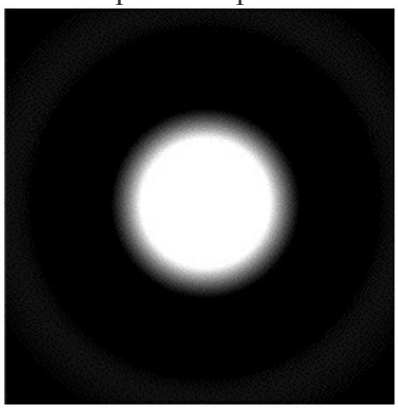



• **catmullrom**: pondère les échantillons à l'aide d'un polynôme cubique Catmull-Rom.

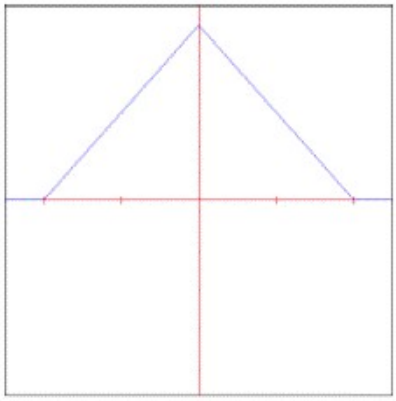



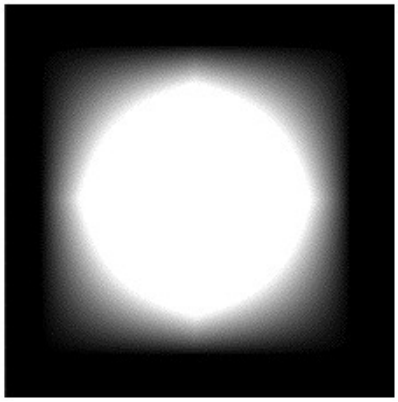
• **bessel**: pondère les échantillons avec une réponse de phase linéaire.





• **triangle**: pondère les échantillons dans la zone de filtrage en utilisant une pyramide.





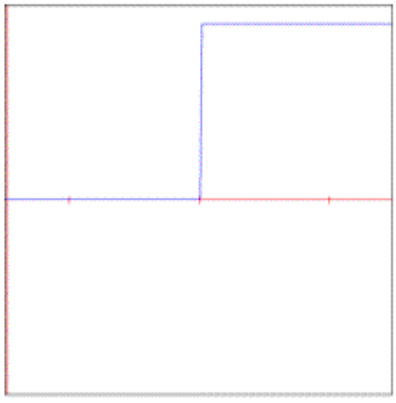
Ces objets sont utilisés comme paramètres pour les objets **noise.value.convolution** et **noise.sparse.convolution**, qui s'attendent à recevoir un objet **filtre** dans le cadre de leur évaluation.

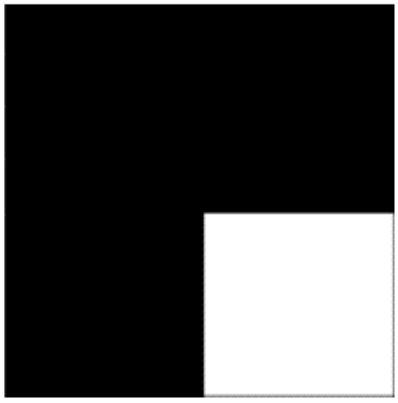
Fonctions de transfert

Les fonctions qui transforment une entrée en une sortie différente sont contenues dans la catégorie des **transferts**. La plupart de ces fonctions ne fonctionnent que sur une seule dimension dans l'intervalle d'unités **0 -1**.

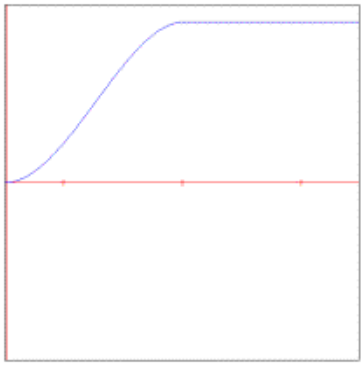
Une brève description de ces fonctions figure dans la liste ci-dessous.

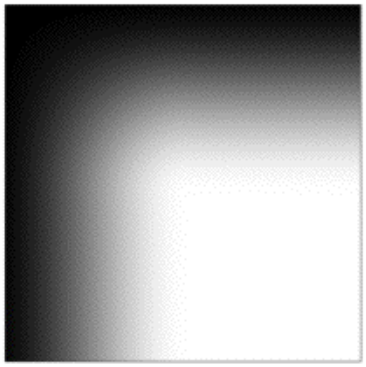
• **step**: toujours **1** si la valeur donnée est inférieure au seuil.



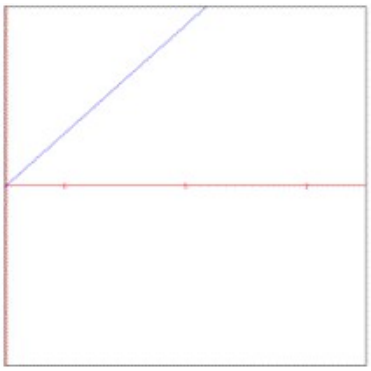


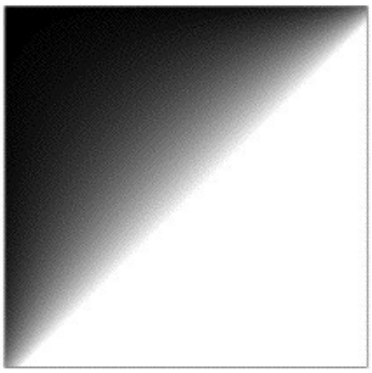
• **smoothstep**: fonction de pas avec lissage cubique aux limites.



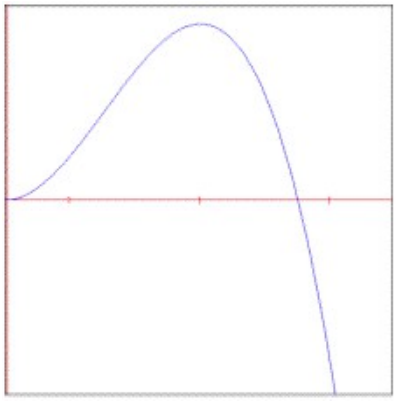


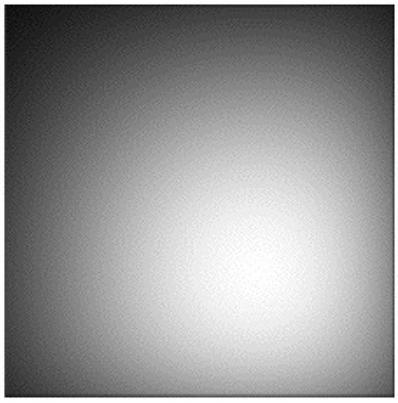
• **bias**: polynôme similaire au gamma mais remappé à l'intervalle unitaire.



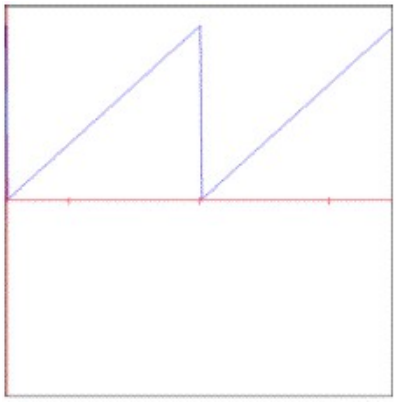


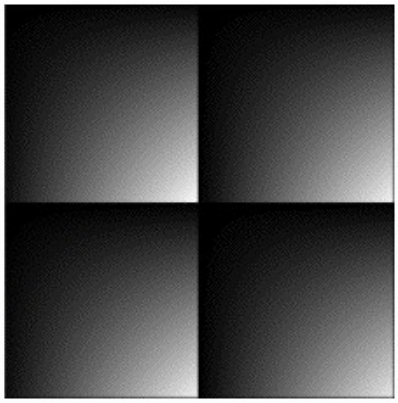
• **cubic**: polynôme générique d'ordre 3 avec coefficients contrôlables.



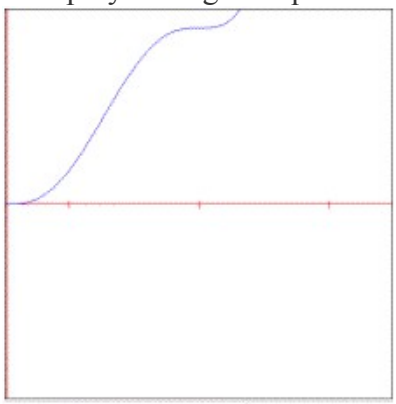


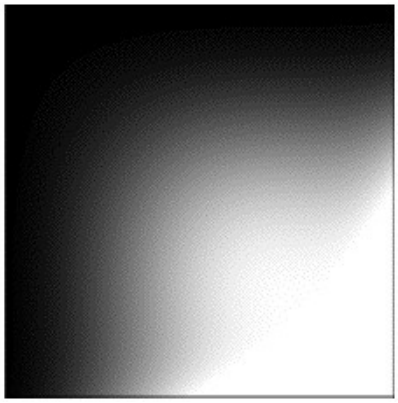
• **saw**: train d'impulsions triangulaires périodiques.



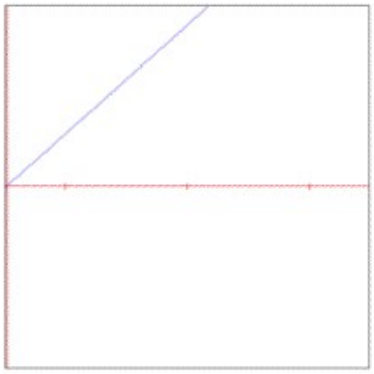


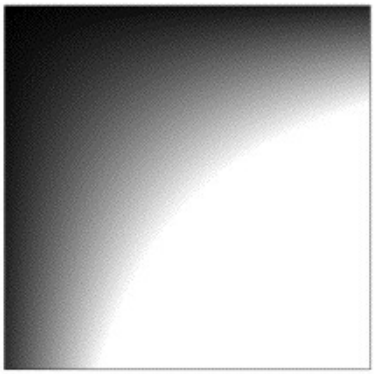
• **quintic**: polynôme générique d' ordre 5 avec des coefficients contrôlables.



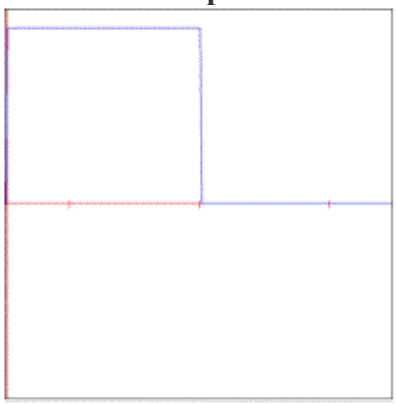


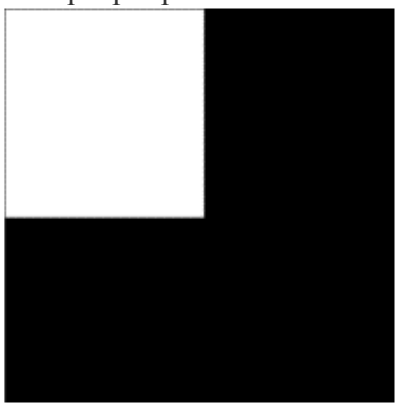
• **gain**: polynôme en forme de S évalué à l'intérieur de l'intervalle unitaire. Remarque: les paramètres par défaut donneront une courbe linéaire au lieu de la forme en S décrite.



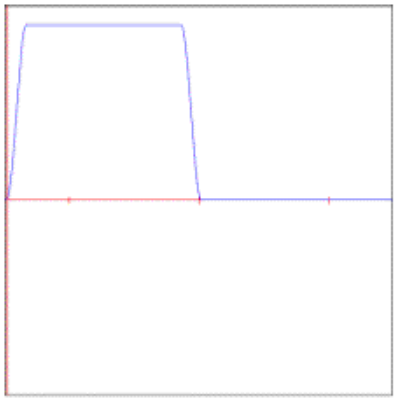


• **pulse**: fonction périodique par pas.

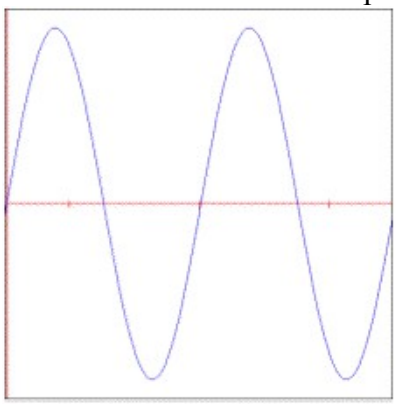


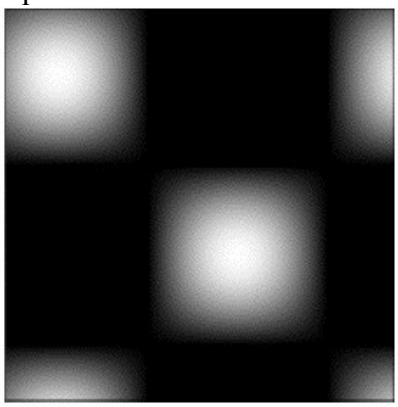


• **smoothpulse**: fonction de pas périodique avec lissage cubique aux limites.



• **sine**: courbe périodique sinusoïdale.





• **linear**: fonction linéaire sur l'intervalle des unités.
• **solarize**: met à l'échelle une valeur donnée si le seuil est dépassé.

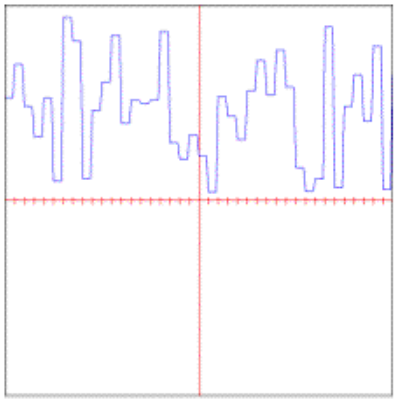
Ces fonctions **transfer** peuvent être utilisées à l'intérieur de plusieurs des objets **noise** pour changer leur fonction de lissage et/ou modifier leur sortie.

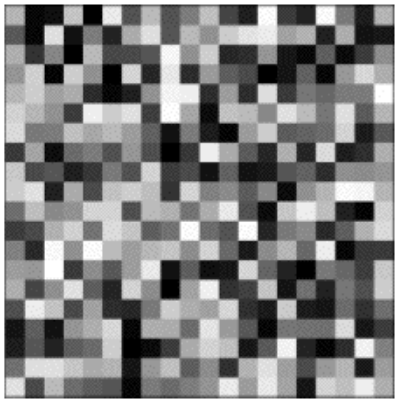
Fonctions Noise

Les modèles stochastiques déterministes (alias fonctions de bruit cohérentes pseudo-aléatoires) sont la pierre angulaire de presque tous les modèles procéduraux. Elles permettent de créer une quantité contrôlable de complexité en ajoutant des détails visuels.

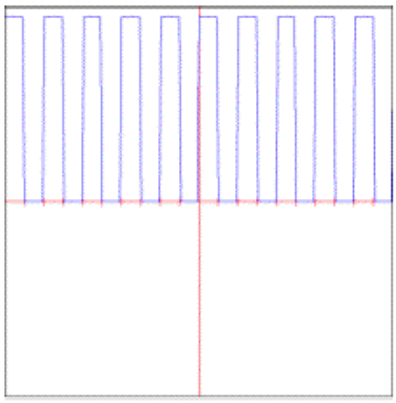
Une brève description de ces fonctions figure dans la liste ci-dessous.

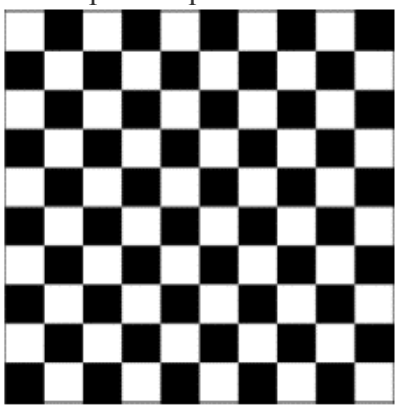
• **cellnoise**: bruit cohérent en forme de blocs.



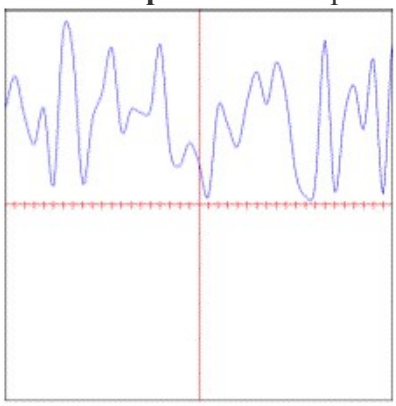


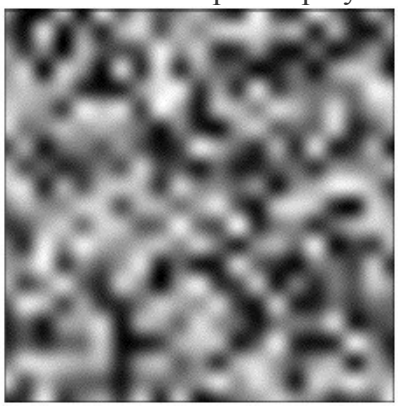
• **checker**: carrés de contrôle périodiques.



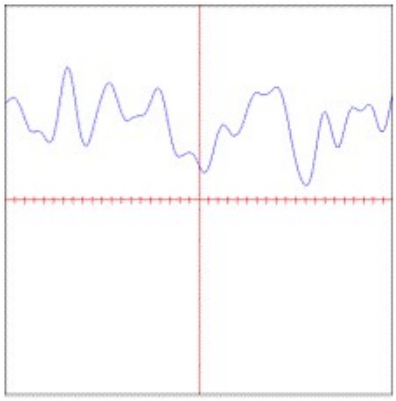


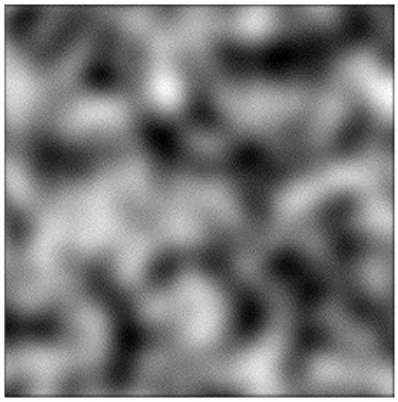
• **value.cubicspline**: valeurs pseudo-aléatoires lissées par un polynôme.



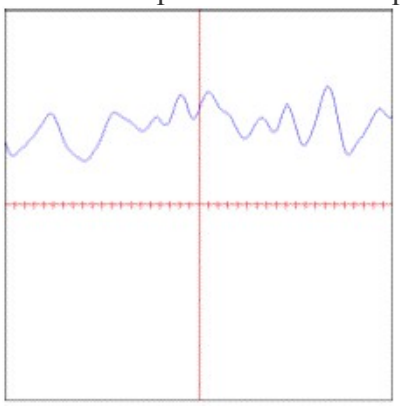


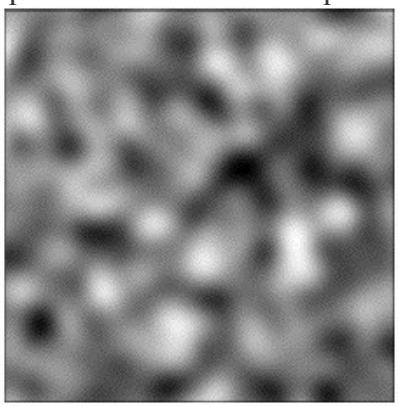
• **value.convolution**: valeurs pseudo-aléatoires filtrées par convolution.



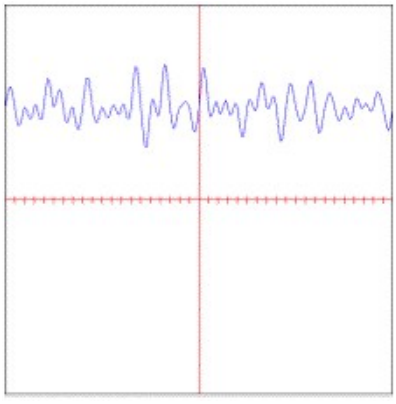


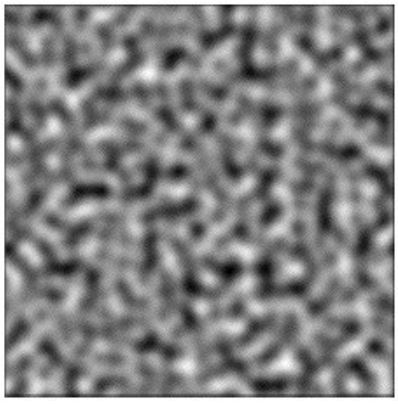
• **sparse.convolution**: points caractéristiques pseudo-aléatoires filtrés par convolution.



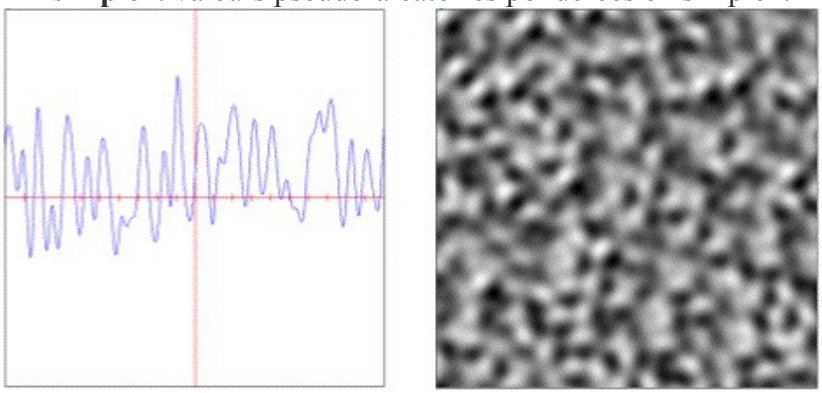


• **gradient**: valeurs interpolées de manière polynomiale pondérées de manière directionnelle.

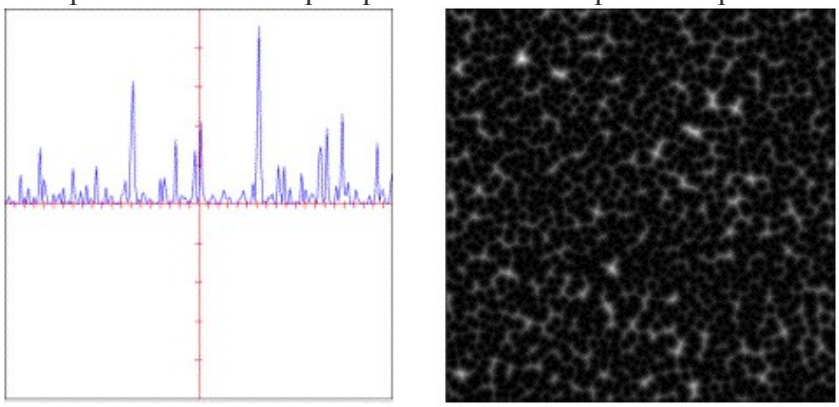




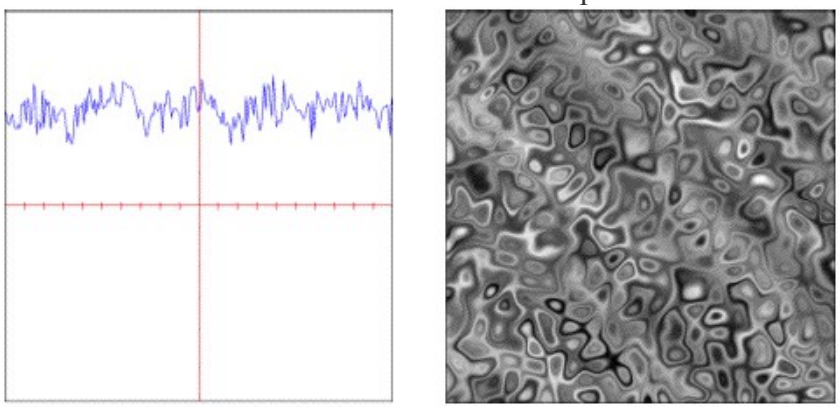
• **simplex**: valeurs pseudo-aléatoires pondérées en simplex.



• **voronoi**: points caractéristiques pseudo-aléatoires pondérés par la distance.



• **distorted**: bruit combinatoire déformé par le domaine.



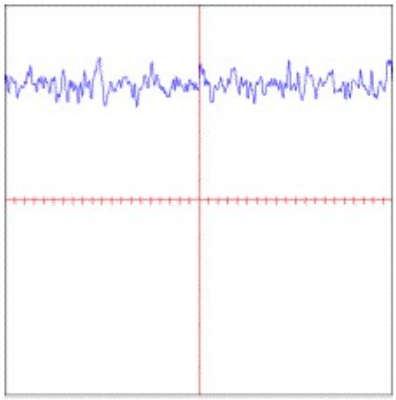
Toutes ces fonctions sont des générateurs à l'exception de l'objet **noise.distorted**, qui est un opérateur binaire et utilise deux fonctions existantes pour son évaluation.

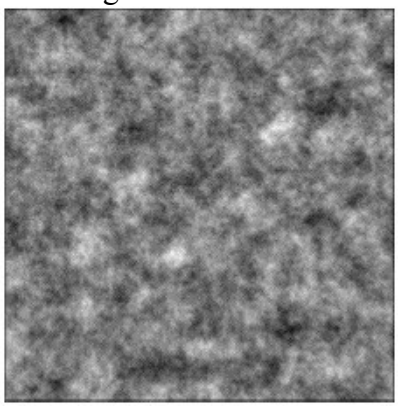
Fonctions fractales

Les fonctions fractales fournissent une forme spécialisée de génération en combinant plusieurs échelles ou octaves d'une autre fonction de base. Ce processus forme l'auto-similarité caractéristique de toutes les fractales.

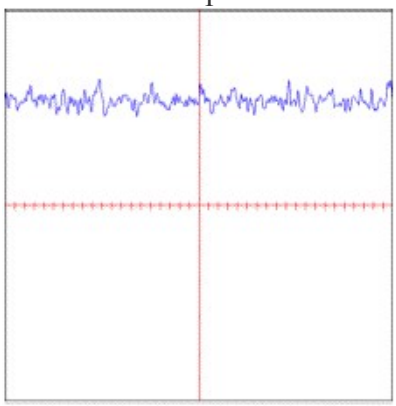
Une brève description de ces fonctions figure dans la liste ci-dessous.

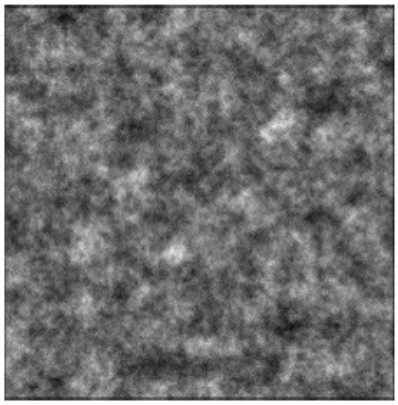
• **mono**: fractale additive avec une similarité globale entre les échelles.



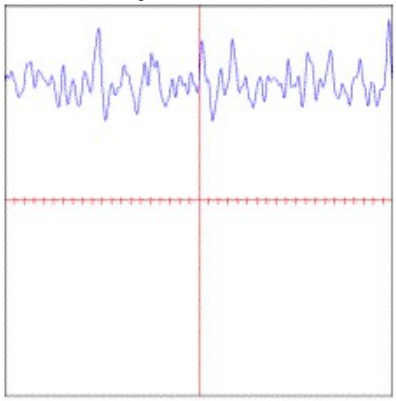


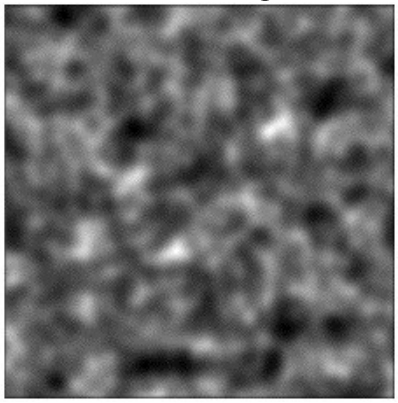
• **multi**: fractale multiplicative avec une similarité variable selon les échelles.



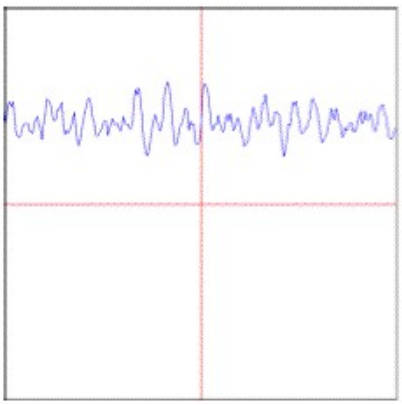


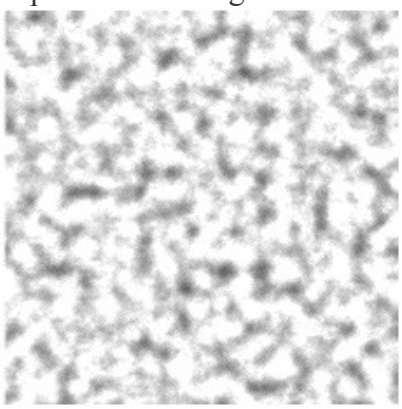
• **multi.hybrid**: une fractale hybride additive et multiplicative.



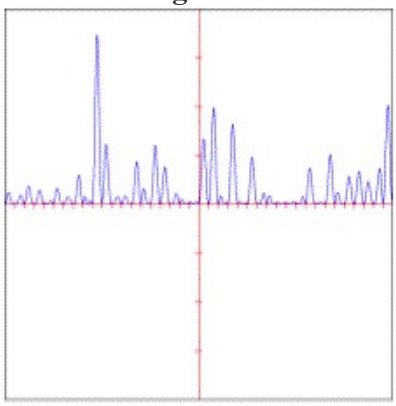


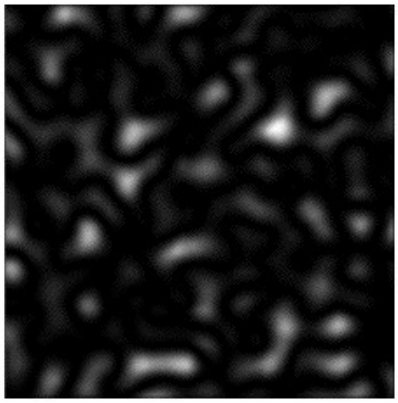
• **multi.hetero**: fractale multiplicative hétérogène.



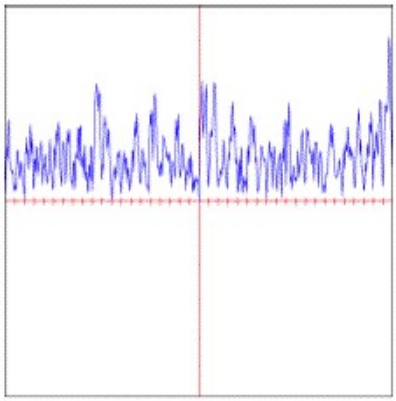


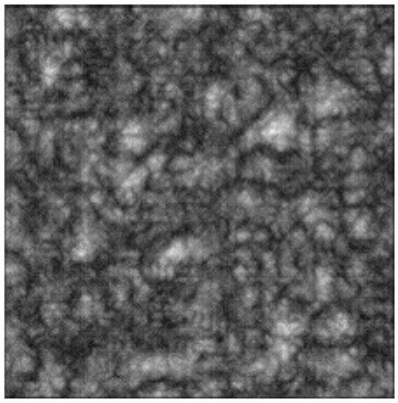
• **multi.ridged**: fractale multiplicative avec des crêtes nettes.





• **turbulence**: mono-fractale additive avec des lignes de crêtes nettes.





Autres attributs et messages

• Sélectionnez la fonction de base **noise.checker** dans l'objet *umenu*.

Notez qu'en plus du message **scale** que nous avons utilisé précédemment, nous pouvons également transformer les coordonnées d'évaluation par **rotation**, translation (via **offset**) et en ajustant leur **origin**.

• Changez les 1ère et 2ème boîtes de *nombres* connectées à **origin** pour modifier l'origine x et y.
• Changez les 1ère et 2ème boîtes de *nombres* connectées à **offset** pour modifier la position d'offset x et y.
• Changez la 1ère boîte de *nombre* connectée à **rotation** pour modifier notre angle de rotation autour de l'axe x.

Remarquez l'effet de la transformation. Notez également la baisse de performance lorsqu'une **rotation** est effectuée - nous obtiendrons toujours de meilleures fréquences d'images si l'attribut **rotation** est laissé à **0** pour chaque dimension de matrice.

> ***Détail technique***: en plus de la génération de coordonnées internes déjà décrite, *jit.bfg* accepte également une matrice d'entrée de coordonnées à évaluer XYZ aux plans 0-2, et la matrice d'entrée doit avoir la même **dim** que la matrice de sortie *jit.bfg*).

• Changez la 1ère boîte de *nombre* connectée à **rotation** à **0** pour désactiver la rotation.
• Cliquez sur le *toggle* connecté à **autocenter** pour activer le centrage automatique.

Si l'attribut **autocenter** est défini sur **1**, les **dimensions** actuelles de la matrice seront utilisés pour placer l'origine au centre de la matrice de sortie, en remplaçant toutes les valeurs déjà définies pour l'origine.

• Cliquez à nouveau sur le *toggle* connecté à **autocenter** pour désactiver le centrage automatique.
• Sélectionnez la fonction de base **noise.gradient** dans l'objet *umenu*.
• Cliquez sur la boîte de *message* **dim 128 128 1**.

Comme mentionné précédemment, toutes les fonctions de base fournies par Jitter peuvent être évaluées sur un nombre quelconque de dimensions. Ce message a changé notre matrice de sortie pour qu'elle soit une matrice 3D, et a défini en conséquence l'évaluation à effectuer dans un espace tridimensionnel. Comme notre affichage est toujours sur un écran 2D, nous n'avons besoin d'évaluer qu'une seule tranche en 3D, et donc notre 3ème **dim** est fixée sur 1.

• Changez la 3ème boîte de *nombre* connectée à **offset** pour modifier la position d'évaluation z.

Remarquez comment nos résultats changent. Nous nous déplaçons maintenant le long de l'axe z comme si nous déplacions en avant/en arrière dans un volume aligné avec l'écran.

• Sélectionnez **float64** dans *l'umenu* connecté au message **precision**.

Le message **precision** peut être utilisé pour modifier la précision d'évaluation interne de l'objet *jit.bfg*. Cela peut être souhaitable si nous avons besoin de résultats plus ou moins précis sans changer le type de matrice de sortie.

• Changez la 3ème boîte de *nombre* connectée à **offset** pour changer la position d'évaluation z.

Remarquez comment la précision plus élevée affecte la fréquence d'images rapportée par l'objet *jit.fpsgui*. Nous devons faire attention à n'utiliser la précision **float64** que lorsque cela est nécessaire.

• Sélectionnez **float32** dans *l'umenu* lié au message **precision**.
• Changez le nombre de plans pour *jit.bfg* de **1** à **3** pour activer la sortie RGB.

En plus de l'évaluation à n dimensions, *jit.bfg* peut générer jusqu'à 32 plans par dimension. Chaque plan est décalé d'une quantité fractionnaire pseudo-aléatoire contrôlée par l'attribut **align**.

• Changez la boîte de *nombre* connectée à **align** pour *jit.bfg*.

Remarquez comment les plans se séparent et deviennent plus visibles à mesure que la quantité **align** augmente.

• Pour voir des exemples plus spécifiques pour différentes combinaisons de fonctions de base, ouvrez le patch d'aide pour l'objet *jit.bf*g et recherchez dans les sub-patchs chaque catégorie de fonction.

> ***Détail technique***: La sortie de *jit.bfg* peut en fait être utilisée comme entrée d'un autre *jit.bfg* pour effectuer une distorsion de domaine, de manière similaire au fonctionnement de **noise.distorted**. Consultez le patch d'exemple dans ***jit-examples/other/jit.bfg.distorter.pat***.

Résumé

L'objet *jit.bfg* nous donne accès à une bibliothèque de fonctions de base et de générateurs procéduraux que nous pouvons utiliser pour définir un modèle procédural pour créer des textures et modifier la géométrie. En interne, *jit.bfg* génère des coordonnées cartésiennes le long d'une grille. Ces coordonnées peuvent être transformées à l'aide des attributs d'**origine**, d'**offset** et de **rotation** correspondants, ou complètement remplacées par une matrice d'entrée contenant des coordonnées d'évaluation.