Travail de fin d'études réalisé en vue de l'obtention du diplôme de Bachelier en Techniques graphiques

LES USAGES DU BRUIT EN INFOGRAPHIE

Promotteur interne :
Mr Nicolas Sottiaux

Ludwig Dejonckheere

2020 - 2021

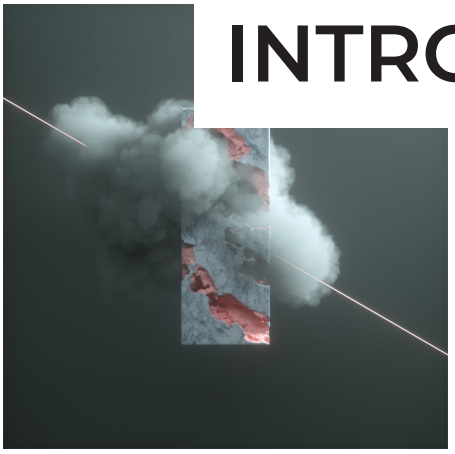
HEH.be
Sciences
et technologies

REMERCIEMENTS

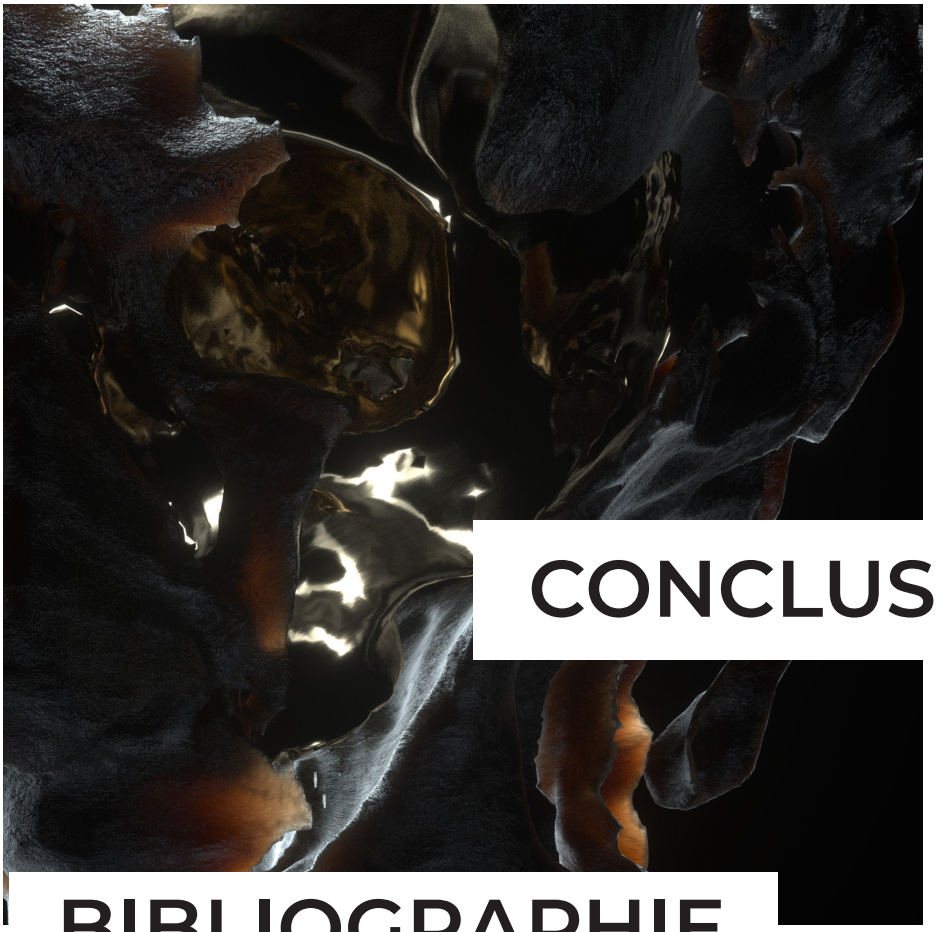
Je tiens à remercier avant tout Monsieur Nicolas Sottiaux, promoteur interne qui m'a assisté durant la réalisation de ce travail de fin d'études. Monsieur Sottiaux a réussi à m'orienter de la bonne manière et m'a redonné confiance lorsque j'ai pu douter de mes productions.

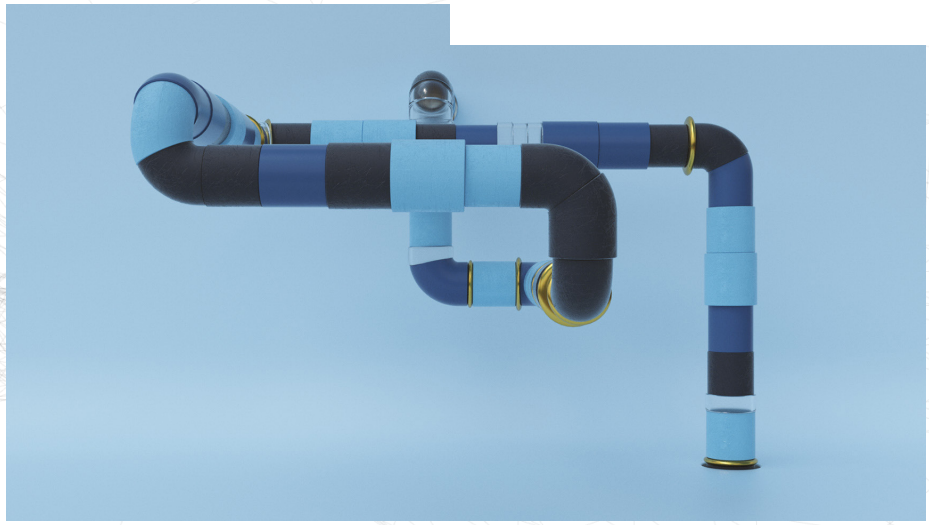
Je remercie également Monsieur Ivan Miller qui a su répondre à mes nombreuses questions concernant ce travail et qui m'a consacré un peu de son temps pour m'éclaircir sur mes interrogations.

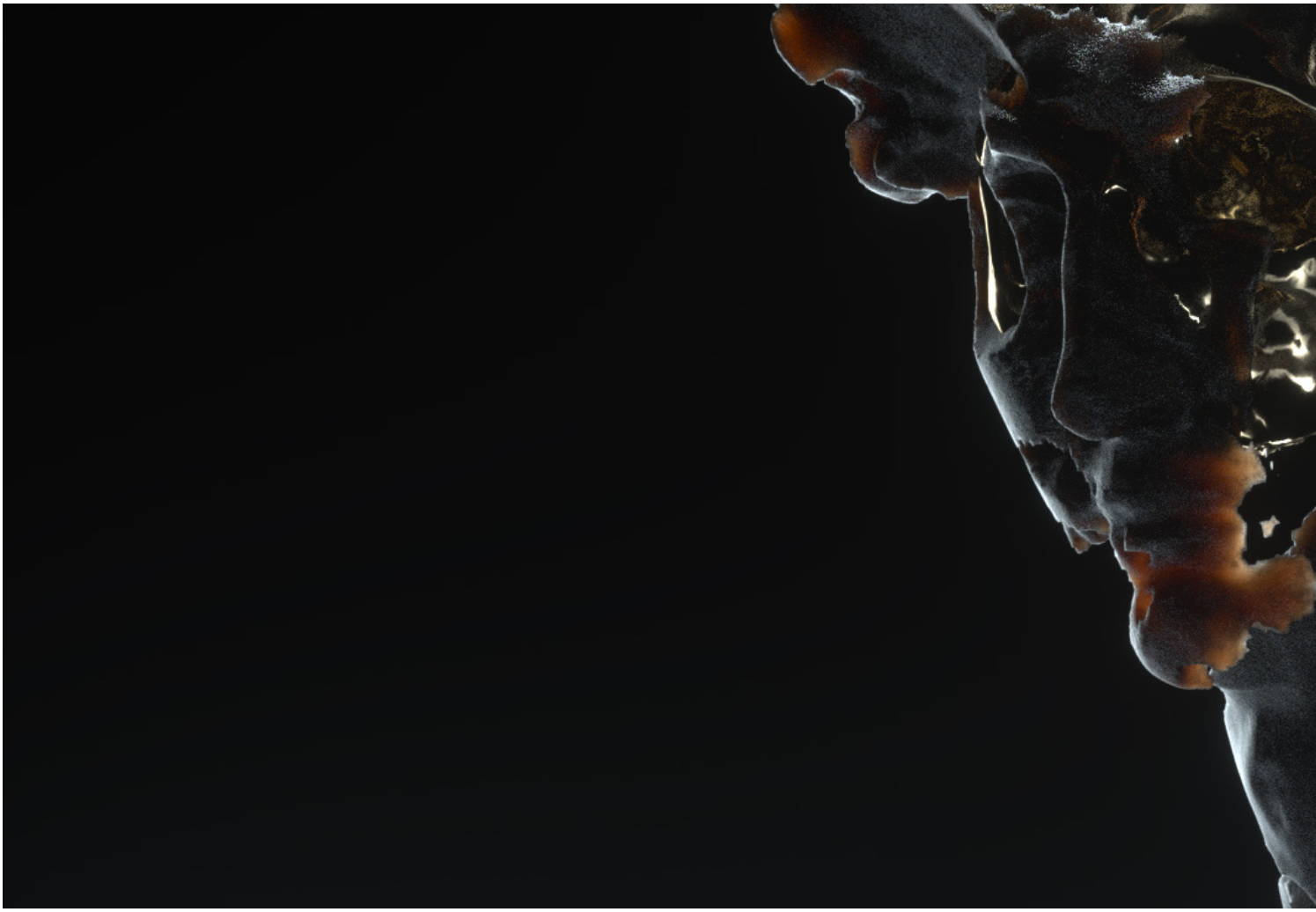
Enfin, je remercie plus globalement la Haute École en Hainaut et la Section Techniques Graphiques, ainsi que le corps enseignant, qui m'ont fait découvrir de nouveaux horizons d'activités plaisants et passionnants.



SOMMAIRE







INTRODUCTION

L'infographie est un terme vaste qui regroupe énormément de compétences. On pense notamment à l'élaborations de logotypes, à la composition graphique, à la création de sites web ou encore de productions audiovisuelles.

Le domaine de la 3D, qui fait partie lui aussi du domaine infographique, est une matière dense qui évolue très vite. Il peut s'agir de dessins techniques et/ou industriels, de compositions graphiques ou encore d'animations. Qu'il s'agisse de motion design (« Motion Graphics » ou « MoGraph ») ou de réalisations pour le cinéma (effets spéciaux ou films d'animations 3D, par exemple).

Parmi toutes ces compétences abordant la 3D, un élément est désormais couramment utilisé pour diverses utilisations : le bruit.

Le bruit est relativement mal connu. Quand on l'évoque, on pense souvent à l'univers sonore. Lorsqu'on envisage le bruit visuel, on a alors tendance à penser au grain photographique. Dans le cadre de ce travail, nous nous concentrons sur un autre type de bruit : le bruit de Perlin (et ses dérivés).

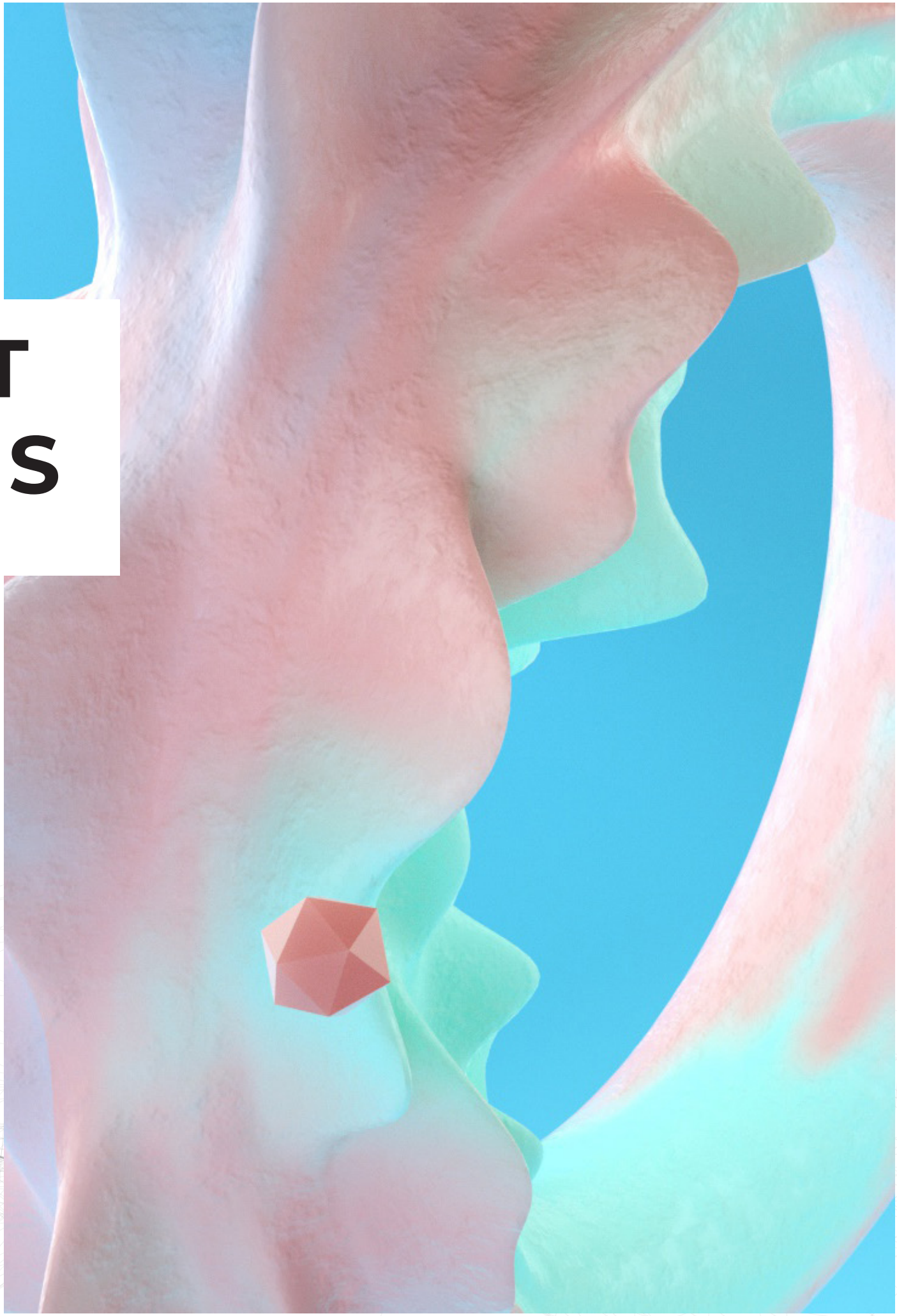
OBJECTIFS E
PRODUCTIO

L'objectif ici est d'expliquer et démontrer les différents usages du bruit dans le monde de la 3D. En commençant par le « displacement », puis le « texturing » et enfin l'animation générée aléatoirement, en passant par la création de volumes organiques..

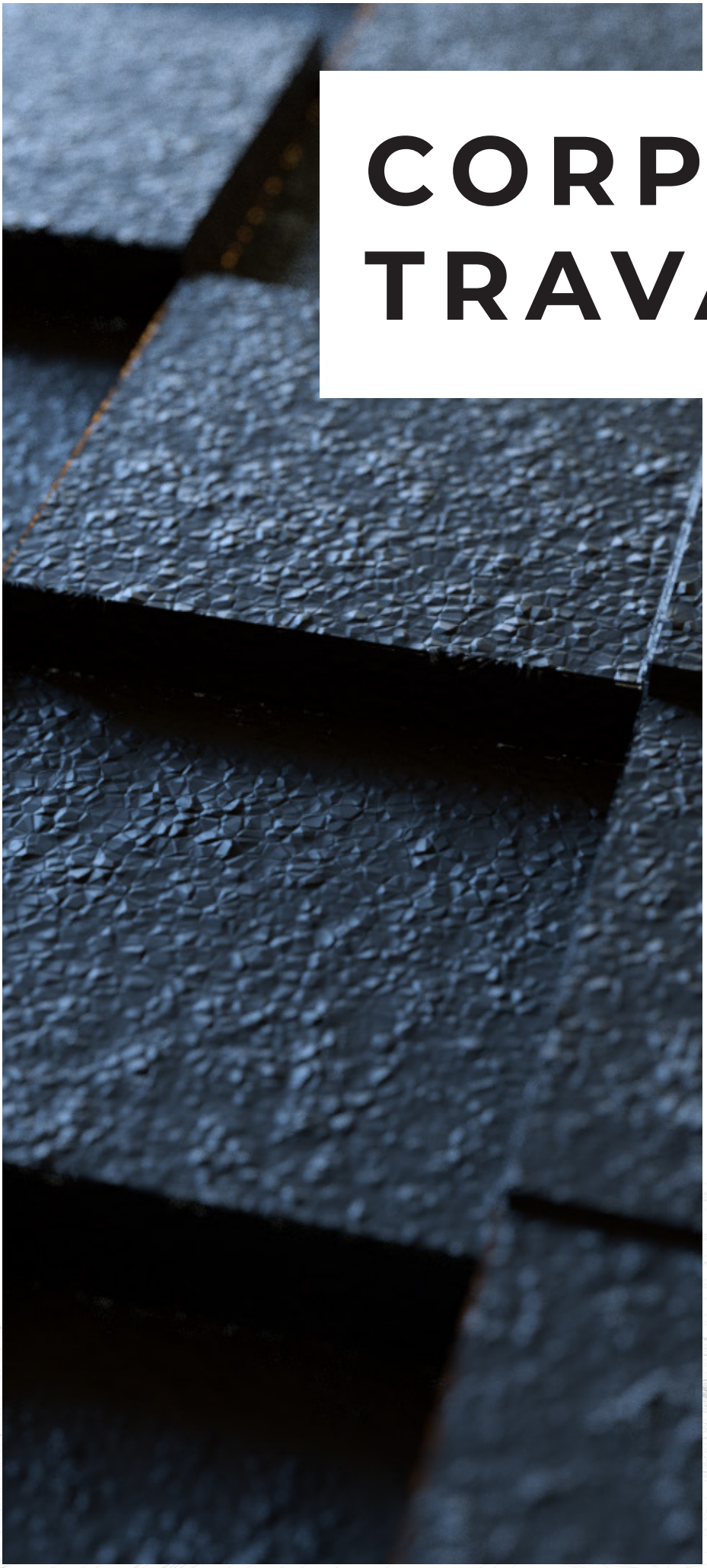
Si ces termes ne vous sont pas familiers, vous les comprendrez parfaitement en parcourant les différentes rubriques de ce rapport.

Enfin, un site spécifiquement dédié à la compréhension et l'utilisation du bruit en 3D est disponible. Ce dernier met en lumière les différentes compositions et œuvres animées élaborées durant ce travail de fin d'études, en incluant une partie pédagogique.

Ce site est avant tout destiné à un public familier à la 3D, plus spécifiquement à Cinema4D et Octane, et qui désire étendre ses connaissances en matière de composition 3D et de motion design.



ET
NS



CORPS DE TRAVAIL

PRÉAMBULE

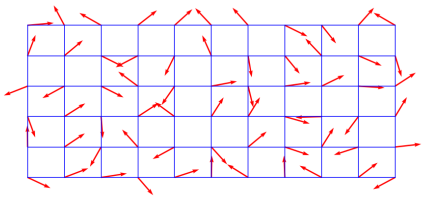
THÉORIE

À l'origine, le bruit de Perlin est une texture procédurale développée par Ken Perlin en 1985, alors qu'il travaillait sur les effets spéciaux du film « Tron ». C'est un bruit de gradient, à ne pas confondre avec le bruit de valeur.

Sans entrer dans des détails trop techniques et en adoptant un langage vulgarisé, un bruit de gradient est un algorithme d'homogénéisation de plusieurs dégradés (= gradients). Ces derniers sont définis par des vecteurs, c'est-à-dire une sorte de flèche qui indique une direction et une intensité.

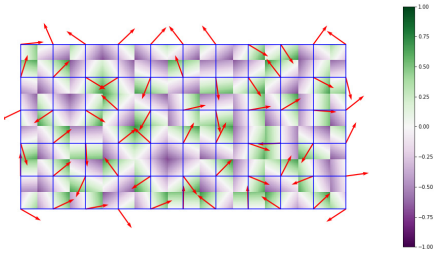
Il y a trois étapes dans la création d'un bruit de gradient :

01



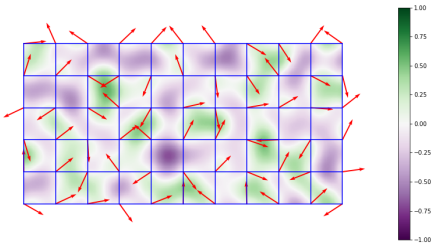
La première étape est la **définition d'une grille** à deux, trois, voire quatre dimensions. Prenons l'exemple d'une grille à deux dimensions. À chaque intersection des axes est attribué un vecteur de gradient aléatoire.

02



La seconde étape de l'algorithme est de calculer **le produit scalaire** de ces vecteurs (=produit des normes de 2 vecteurs par le cosinus de l'angle formé) .

03



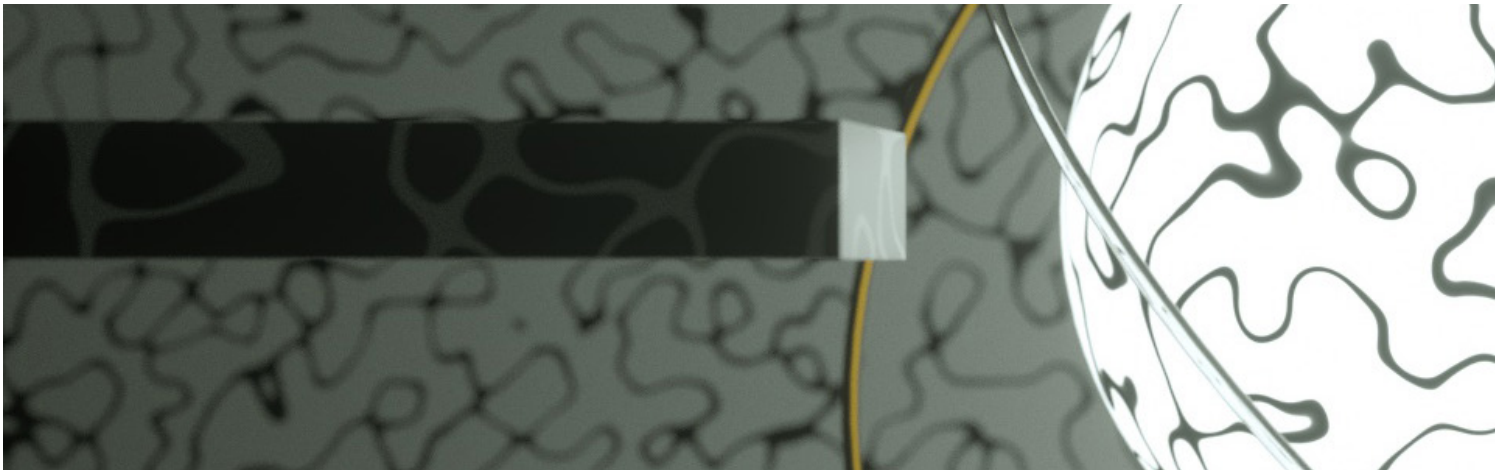
Enfin, la dernière étape est **l'interpolation** entre les produits scalaires calculés aux nœuds des cellules de la grille.

Le bruit apparait alors concrètement.

Le bruit de Perlin génère une texture procédurale (c'est-à-dire définie par un algorithme) pseudo-aléatoire, dans le sens où elle fonctionne sur un système de « seed » (graine en français). Sans cette valeur, il ne peut pas y avoir de génération. Cette graine génère un signal qui est à l'origine du bruit. À valeur différente, génération du bruit différente et donc schéma différent.

Du fait que le bruit se base sur un algorithme mathématique, son grand avantage est qu'il est tout à fait manipulable et permet une infinité de possibilités.

Nous nous concentrerons ici sur les différents bruits que propose le logiciel Cinema4D qui ne compte pas moins de 25 bruits différents et modulables. Ces bruits sont des dérivés du bruit de Perlin : leurs algorithmes varient mais ils fonctionnent selon le même principe.



LOGICIELS

Le premier logiciel utilisé ici est Cinema4D, un logiciel de création 3D développé par la firme Allemande Maxon en 1990. C'est un logiciel complet concernant la 3D puisqu'il comprend les outils de modélisations standards, des fonctionnalités de sculpture 3D, la création de matériaux, les outils nécessaires pour créer des rendus et enfin une bibliothèque exhaustive d'effets d'animations 3D : le « MoGraph ». Cinema4D est connu pour être adapté aux débutants se lançant dans le monde de l'animation 3D. Il est également tout à fait adapté et performant pour les motion designers professionnels.

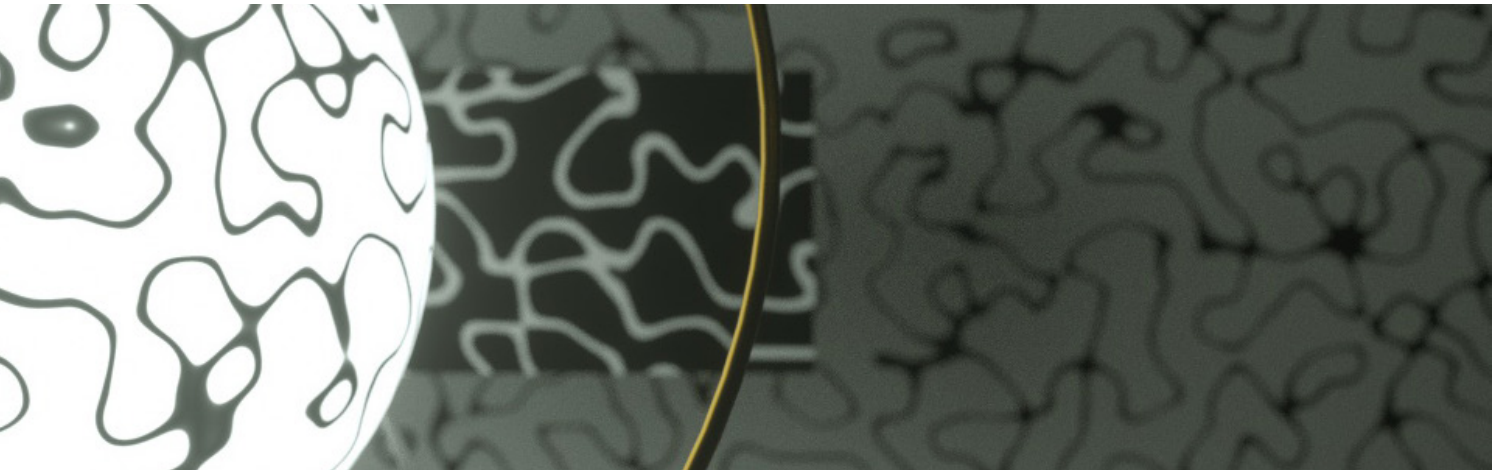
Le second logiciel est utilisé en tant que plugin de Cinema4D : Octane Render qui est un logiciel de rendu 3D développé en 2012 par la société néo-zélandaise OTOY. Il a plusieurs avantages qui font de lui un moteur de rendu très utilisé par les motion designers.

Premièrement, il s'agit d'un moteur de rendu en temps réel. Il n'y a donc pas besoin de lancer un rendu et de devoir attendre que celui-ci soit entièrement complété pour apprécier le résultat final. Ici, le rendu de la scène se fait directement dans une fenêtre séparée.

Deuxièmement, il utilise pleinement le processeur graphique (GPU), c'est-à-dire la puissance de calcul de la (des) carte(s) graphique(s) de l'ordinateur. Ce n'était pas le cas de la plupart des moteurs de rendu il y a encore quelques années. En effet, ces derniers s'appuyaient sur le processeur principal des ordinateurs, ce qui tendait à rendre le temps de rendu beaucoup plus long.

Ces deux facteurs sont déjà de bonnes raisons d'utiliser Octane, surtout si le processeur graphique est de dernière ou avant-dernière génération, étant donné leur puissance de calcul extraordinaire.

Enfin, parmi la palette d'avantages qu'offre Octane, il y a le mode « Pathtracing ». C'est une technique de « lancer de rayons » (ou « Ray Tracing » en anglais) qui détermine l'illumination de la scène 3D en fonction du rebond des rayons de lumières sur les différentes surfaces. L'image est d'abord un brouillard de pixels qui s'affine avec le temps. Cette technologie tend à créer des rendus très « réalistes ».

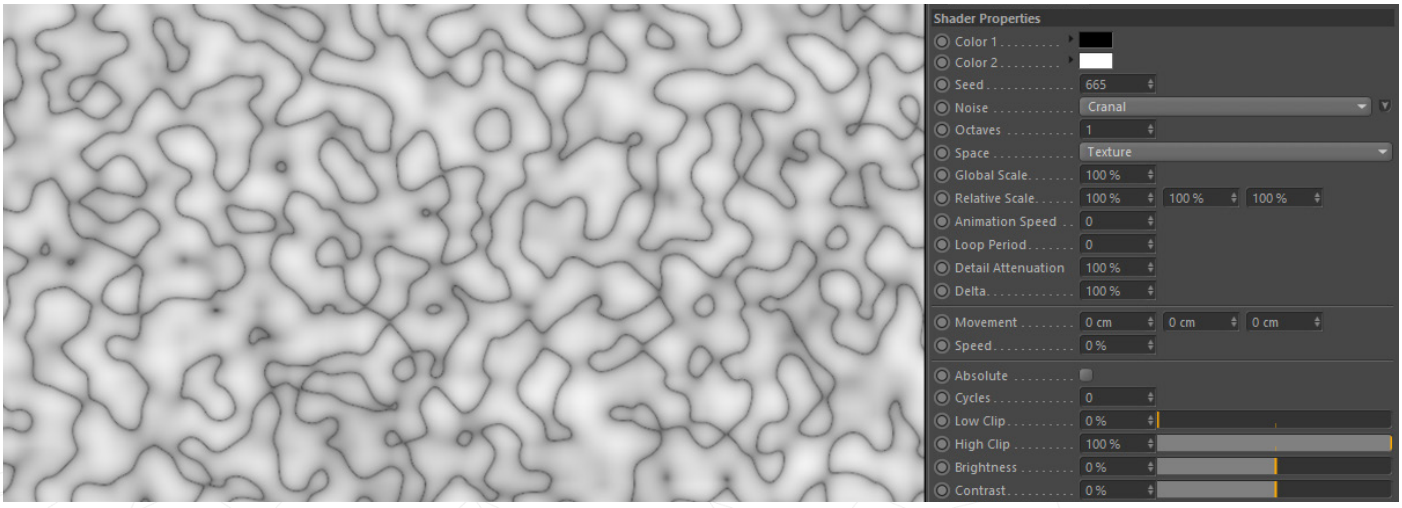


PARAMÈTRES DU BRUIT DANS CINEMA4D

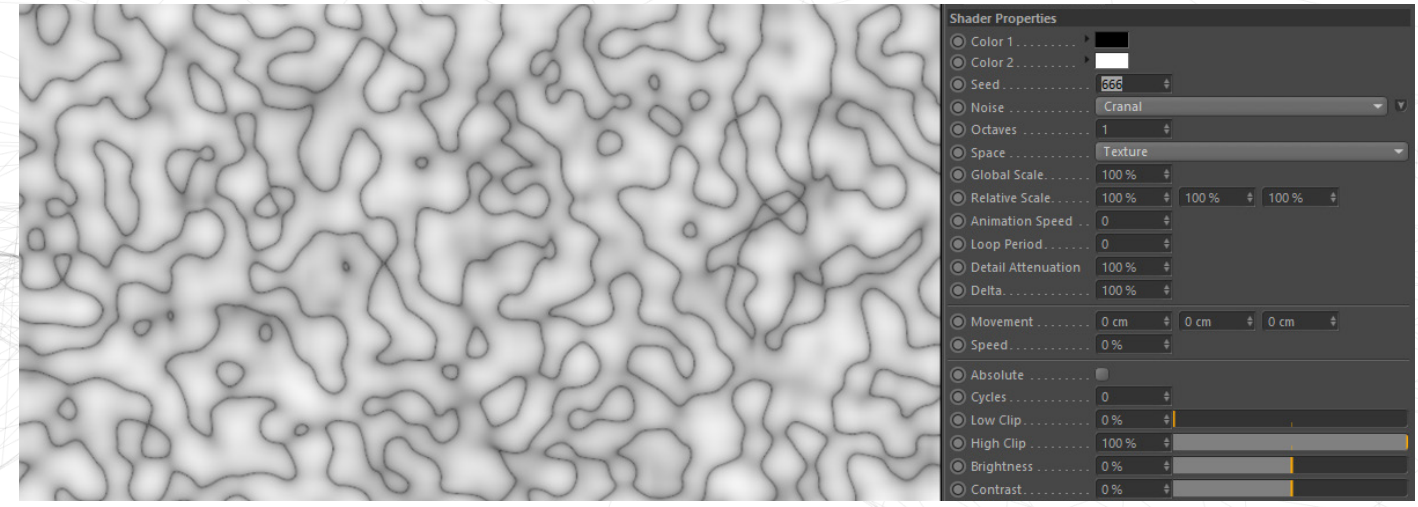
Avant de démontrer les applications du bruit, il est primordial de définir et comprendre les différents paramètres qui le composent. Ces paramètres sont les piliers qui définissent l'aspect même du bruit et qui caractérisent sa complexité.

Le bruit « Cranal » est ici pris comme exemple pour bien distinguer les nuances lorsqu'un paramètre est modifié.

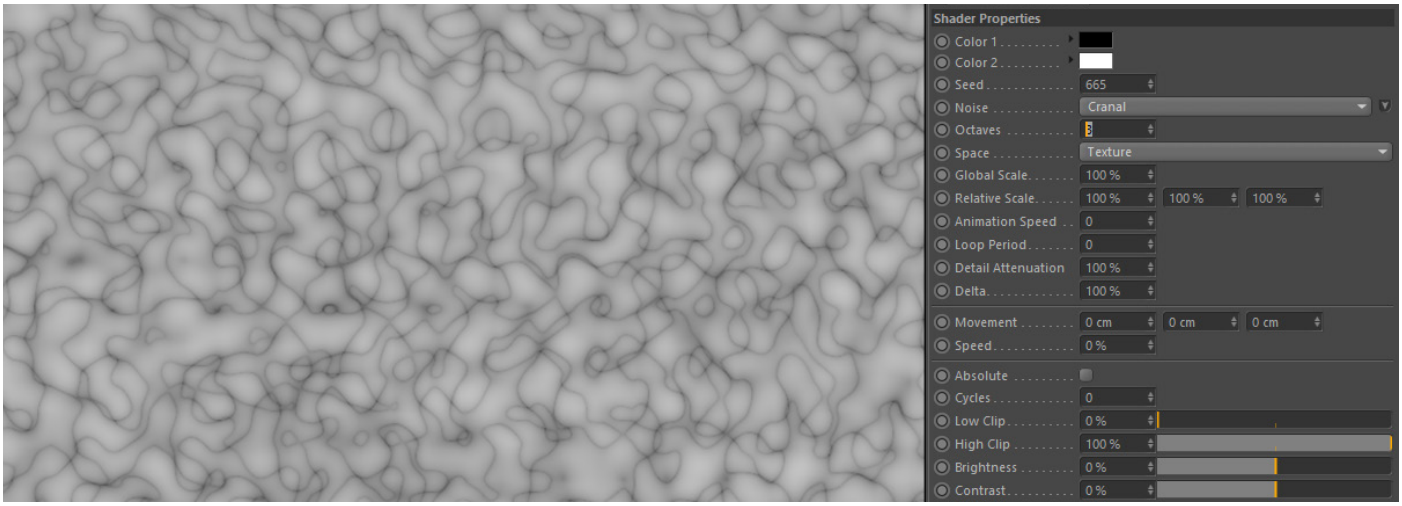
Voici l'aspect 2D qu'affiche le bruit par défaut, avec la fenêtre de paramètres se situant sur la droite.



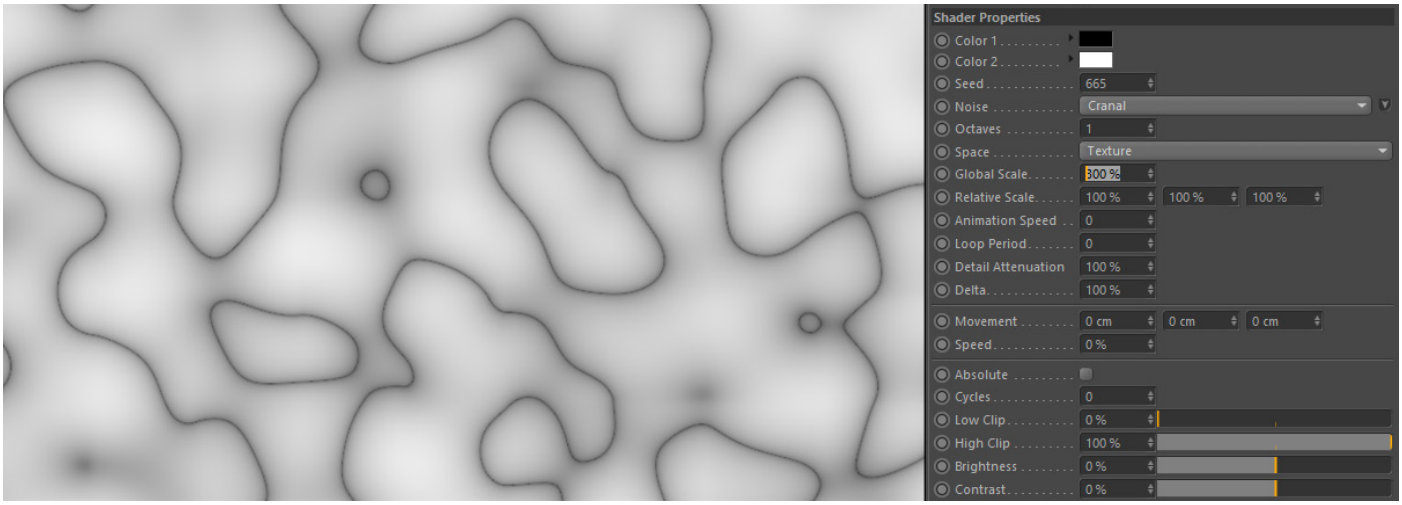
Outre les teintes des couleurs qui sont modifiables, le premier paramètre est la graine (« seed »). Comme énoncé en préambule dans la théorie, la graine est ce qui génère le bruit. Ce dernier présente une génération différente pour chaque graine.



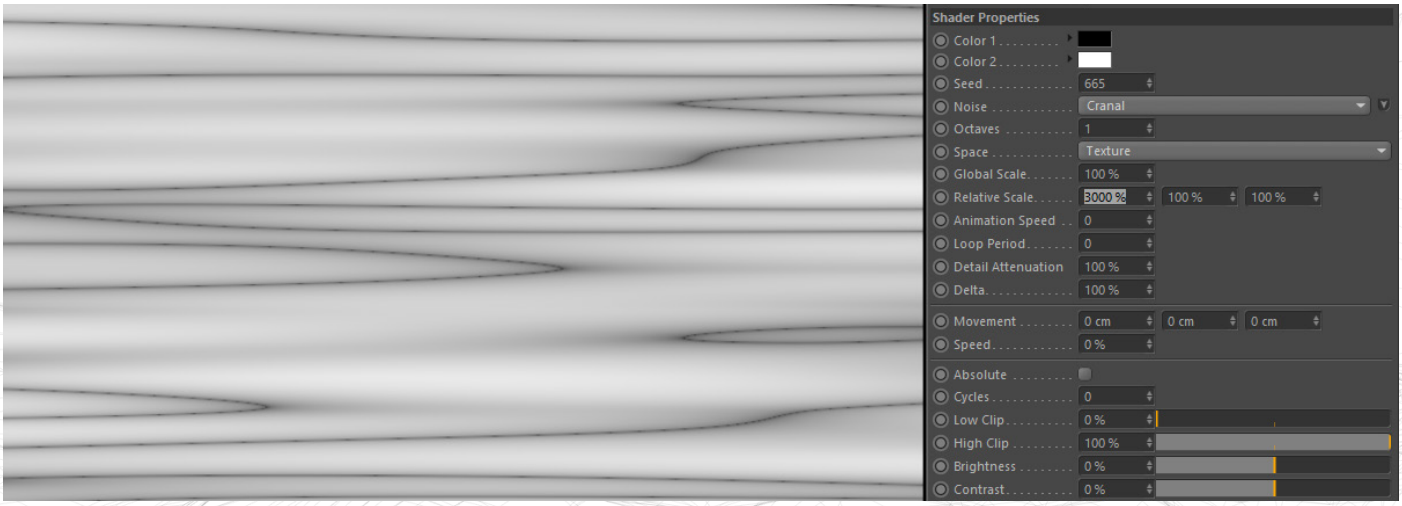
Le second levier sont les octaves (le « Noise » sera vu plus en détails). Elles jouent ici un rôle de complexification du bruit. Plus le nombre d'octaves est élevé, plus le bruit est détaillé, complexe.



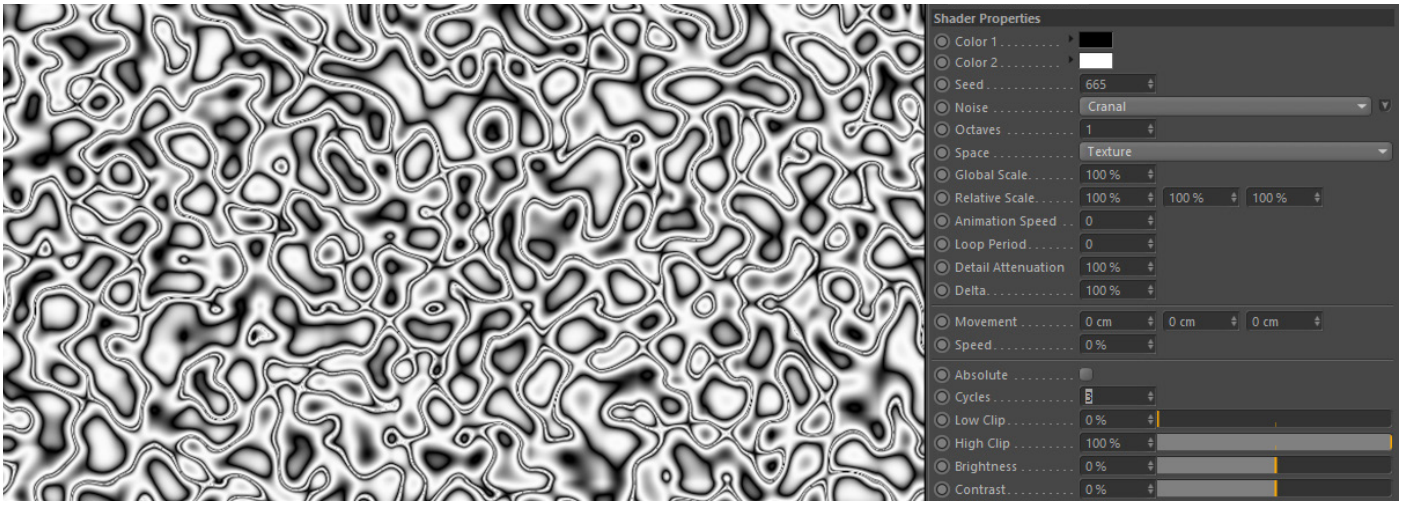
L'échelle globale est le paramètre suivant. Il ne joue ici que le rôle d'échelle standard. Le bruit peut être agrandi ou rétréci.



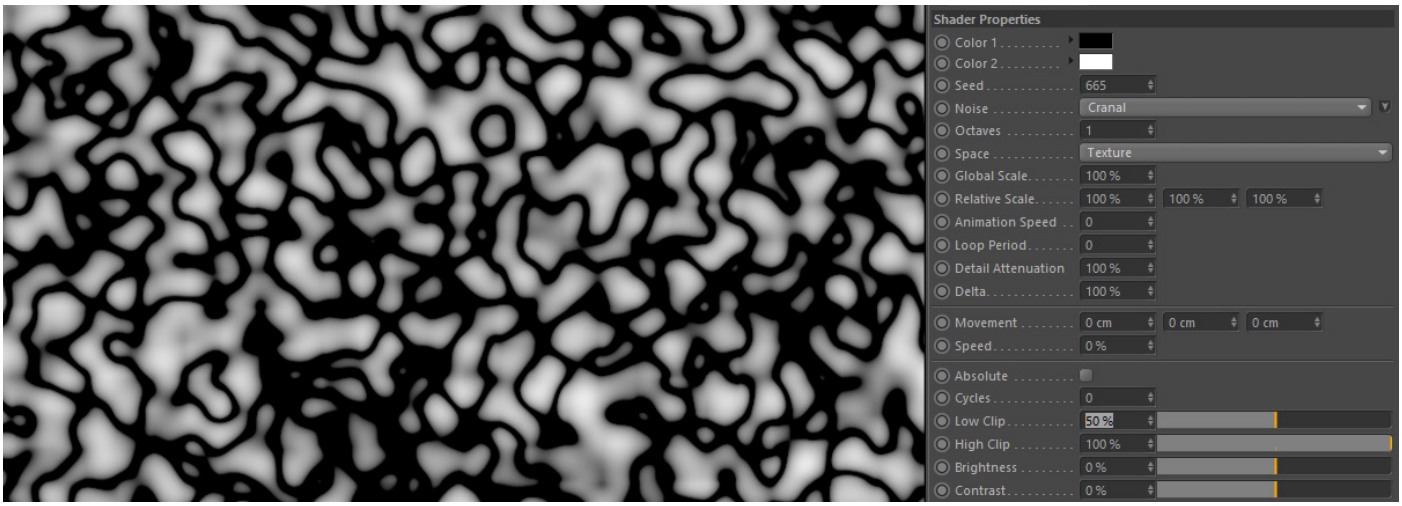
Les échelles relatives aux axes 3D que sont X, Y et Z suivent l'échelle globale. Puisqu'il s'agit ici d'une texture en deux dimensions, un des axes n'est pas effectif.



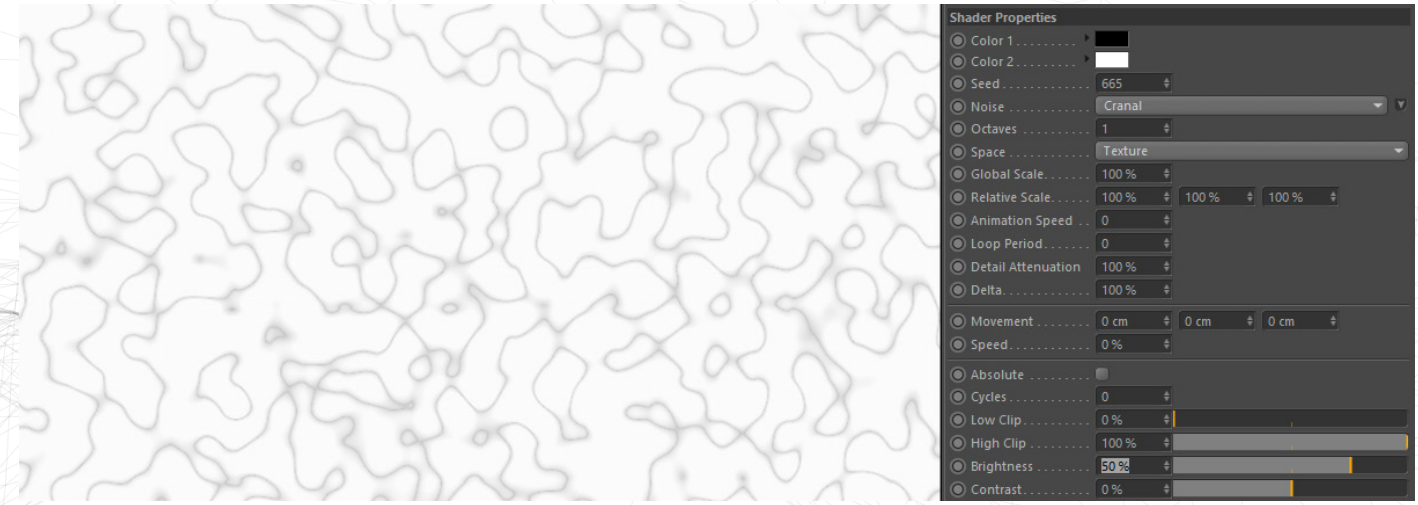
Le paramètre suivant intéressant est ici le cycle. Le cycle entraîne une sorte de répétition du bruit.



Viennent ensuite les paramètres de « Low Clip » et « High Clip » qui agissent un peu à la manière du gamma. En d'autres termes, le clip bas étend ici la zone noire et le clip haut étend la zone blanche. Si la valeur est poussée à son maximum, le bruit devient finalement soit complètement noir soit totalement blanc, en fonction du paramètre choisi.

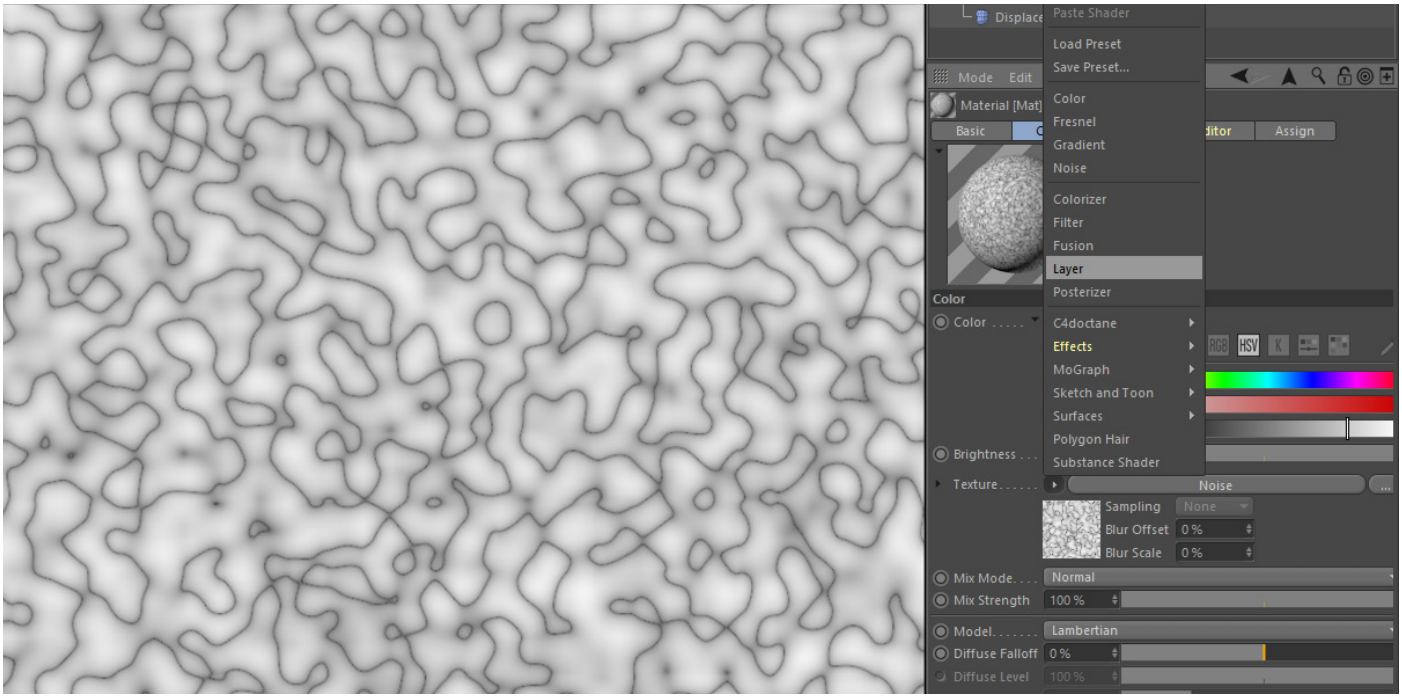


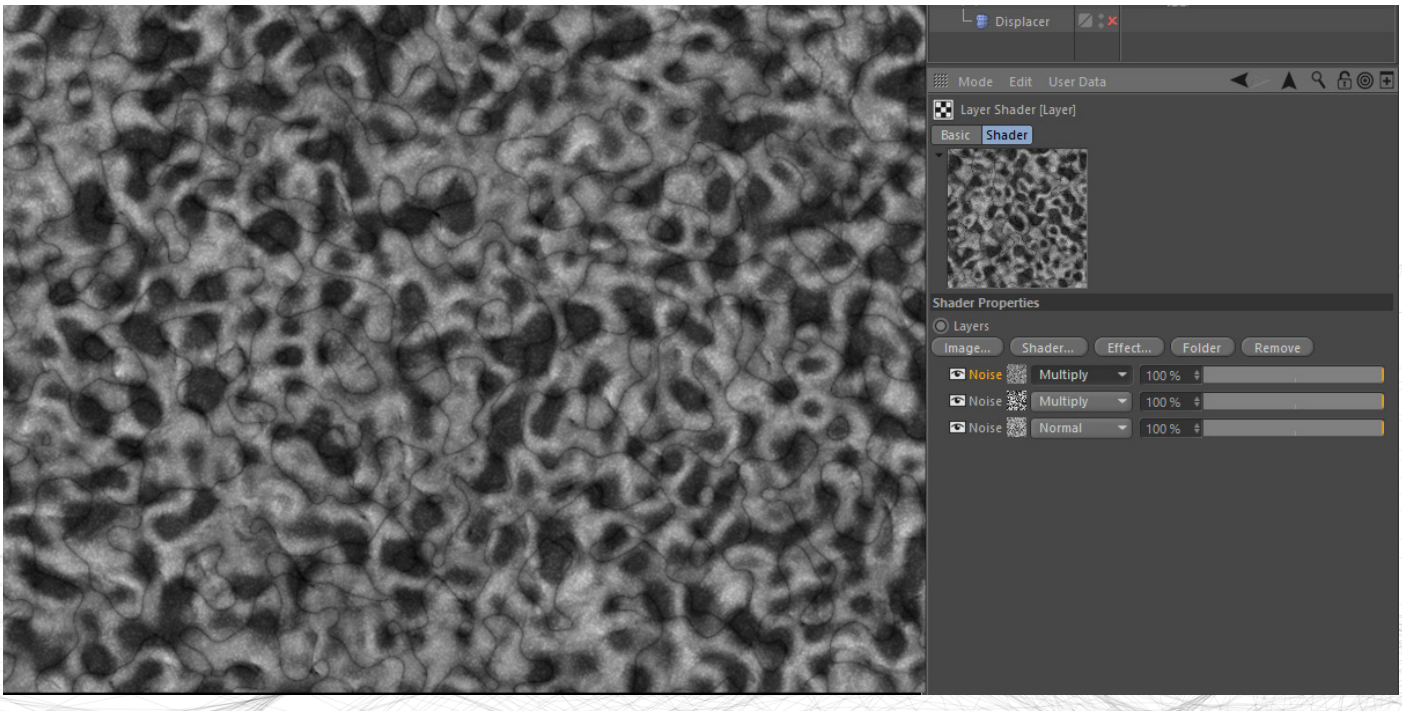
Enfin, les derniers leviers utiles sont ici la luminosité et le contraste, avec la particularité de pouvoir encoder des valeurs négatives, à la manière de Photoshop.

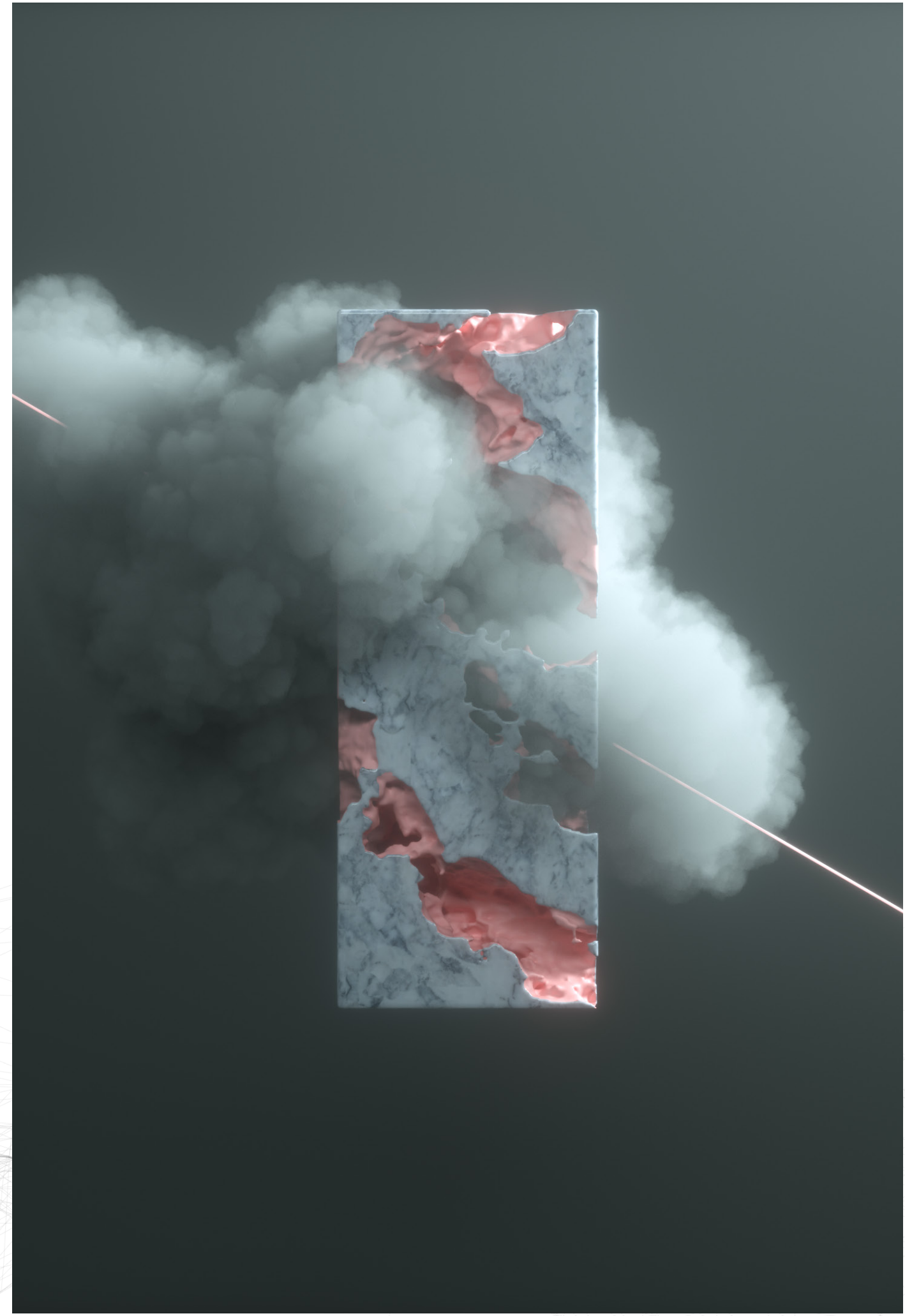


Une fois que tous ces facteurs sont déterminés, le bruit se présente alors comme une palette infinie de formes générées aléatoirement, modulables et surprenantes.

À noter qu'il ne s'agit ici que de simples démonstrations des paramètres et ce, sur un seul bruit parmi une gamme de 25 bruits différents proposés par Cinema4D. De plus, tout comme au sein du logiciel Photoshop, ces bruits peuvent être utilisés sous forme de calques qui sont alors additionnables avec des modes de fusions.







OUVRAGES PRATIQUES

DISPLACEMENT

Qu'est-ce que le displacement ?

Le « displacement », littéralement « déplacement » en français, est l'action de générer du volume sur base d'une texture 2D. Ce volume prend forme en fonction de tons noir et blanc que composent la texture. Plus la teinte de la texture se rapproche du blanc, plus le displacement est à son maximum. A contrario, plus la teinte se rapproche du noir, plus le displacement ne prend pas effet.

Il peut s'agir simplement d'une couleur unie, d'un dégradé, d'une image, etc. Dans ce cas-ci, il est question d'appliquer un bruit en texture.

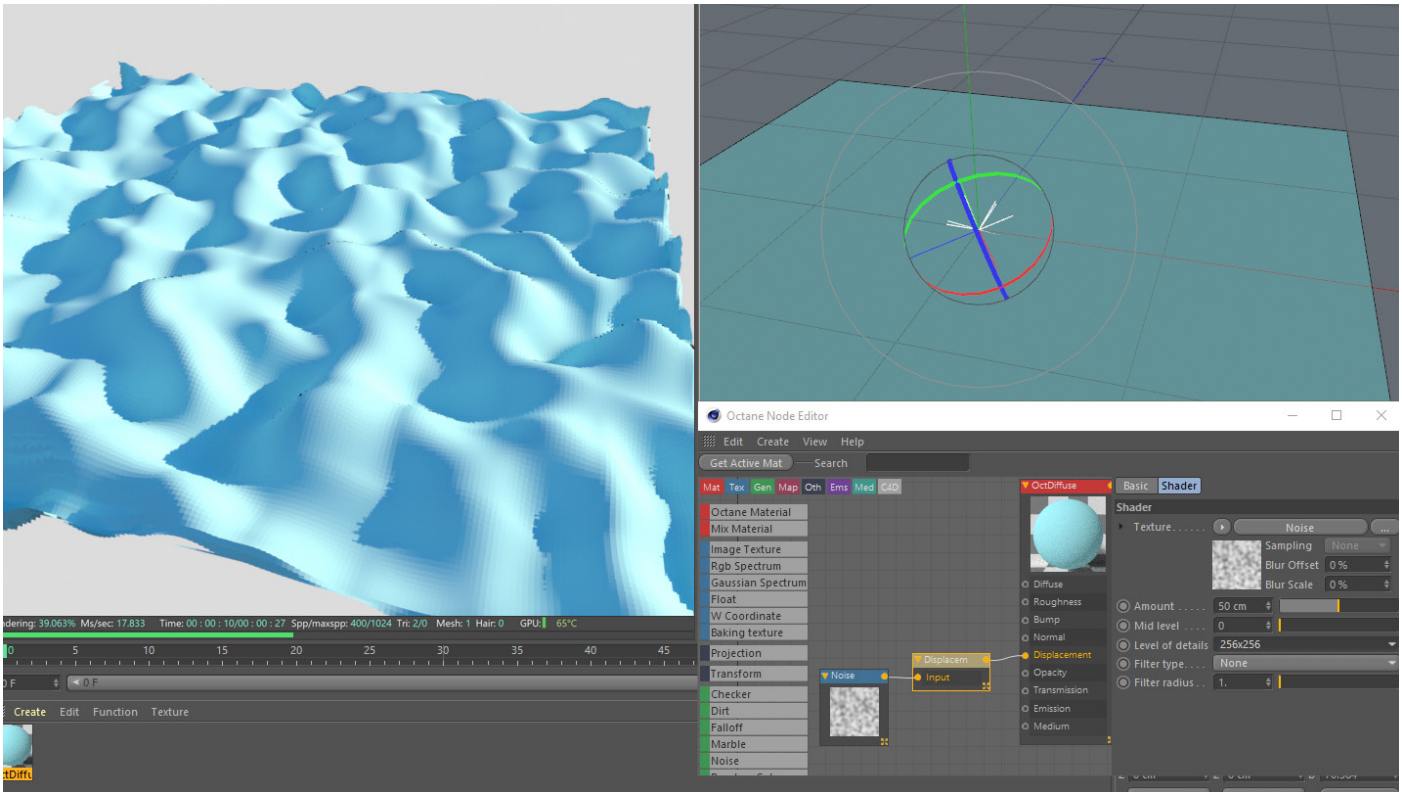
Distinction entre Cinema4D et Octane

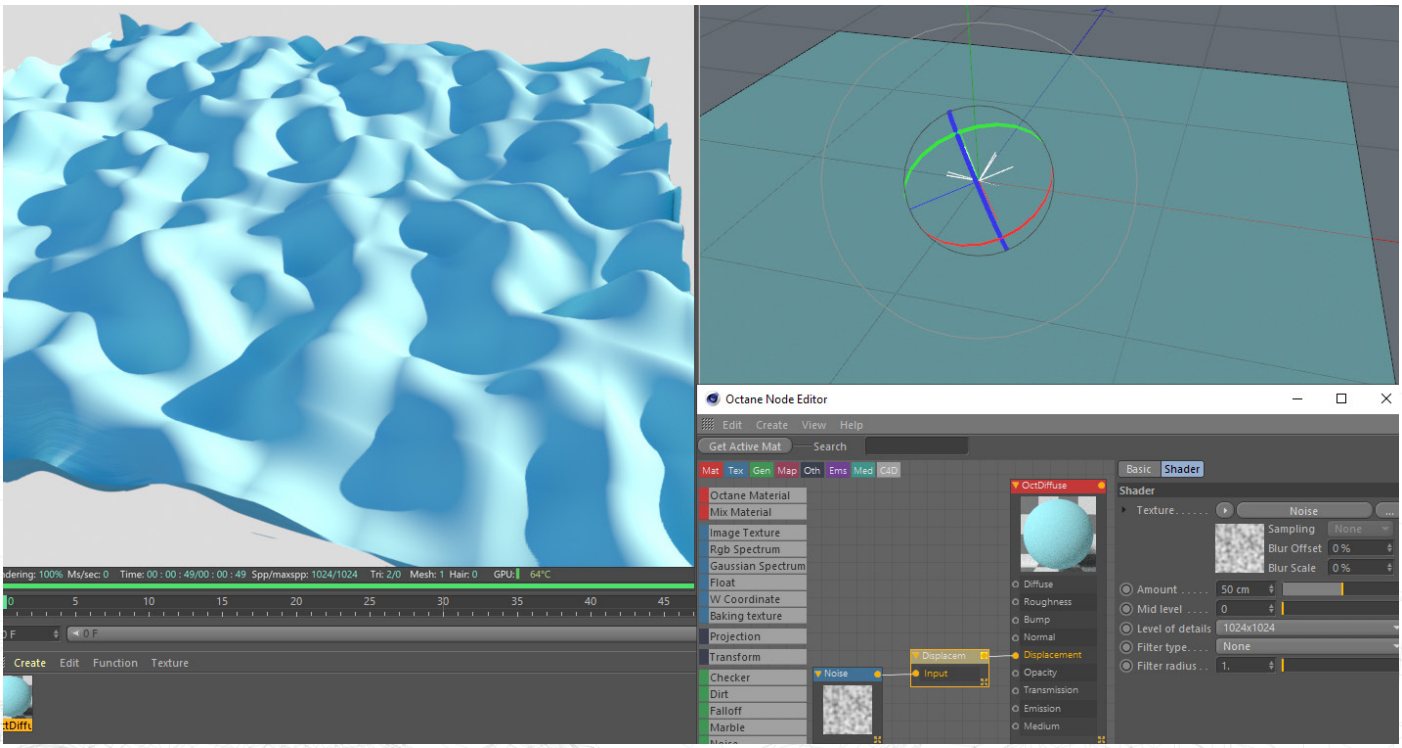
Cinema4D propose son lot de bruits, la plupart du temps utilisés dans un déformeur de géométrie (« displacer ») lorsqu'il s'agit de lui donner un volume aléatoire en fonction du bruit. Cependant, le bruit de Cinema4D est aussi utilisable lors de la création d'un matériau Octane.

La différence ici est que si la déformation est établie directement dans le matériau via le calque de déformation, c'est la mémoire graphique qui travaille, et qui tend à ralentir le logiciel.

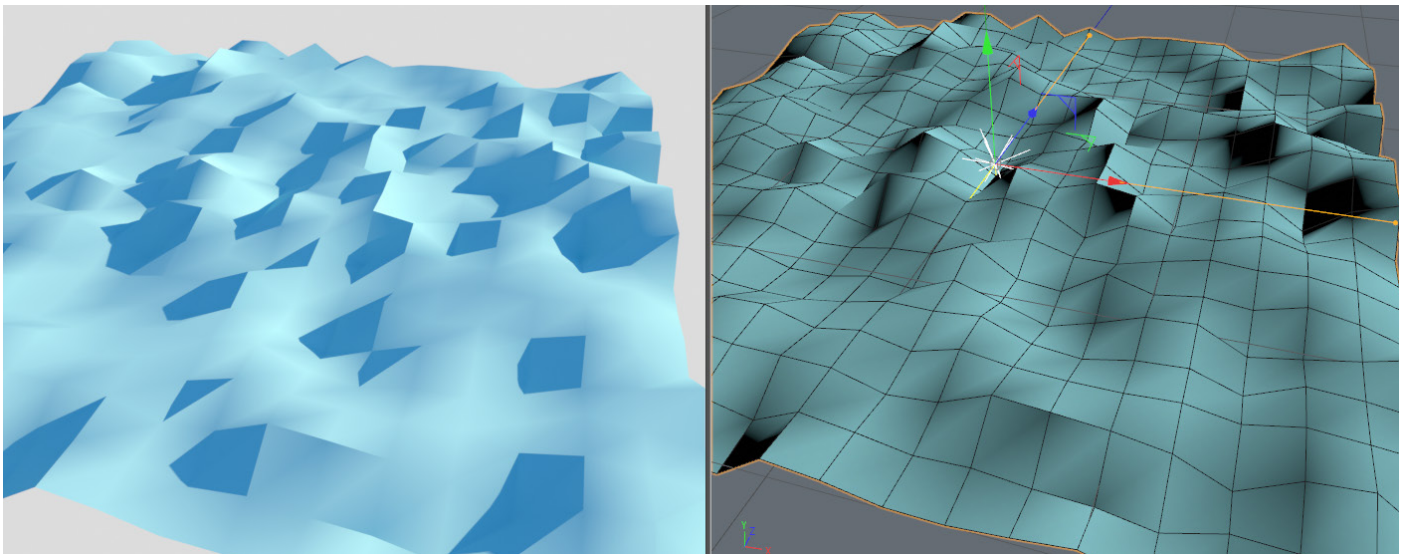
De plus, le displacement est considéré comme matériau et non comme une simple géométrie. Si ce même matériau est appliqué à différents objets 3D, alors ils subiront tous la déformation suivant le matériau.

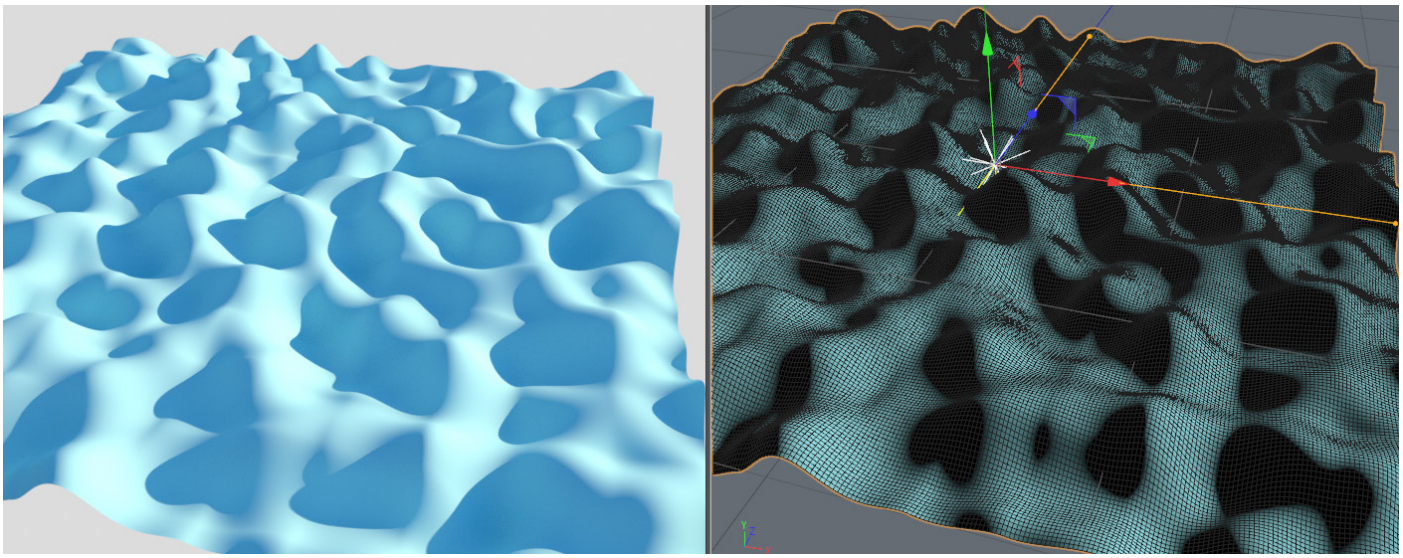
Néanmoins, la déformation sera de qualité supérieure si le bruit est inséré directement dans le matériau. Ceci s'explique par le fait que plusieurs définitions du bruit sont proposées au sein du matériau. Il peut très bien être de moindre qualité en 256x256 px, comme être très précis en 8192x8192 px. Plus la définition est élevée, plus le matériau occupera de la mémoire graphique et plus le logiciel sera lent. À noter que comme il est question ici de matériau, la géométrie propre à l'objet ne change pas, c'est bien dans le rendu que la déformation a lieu.

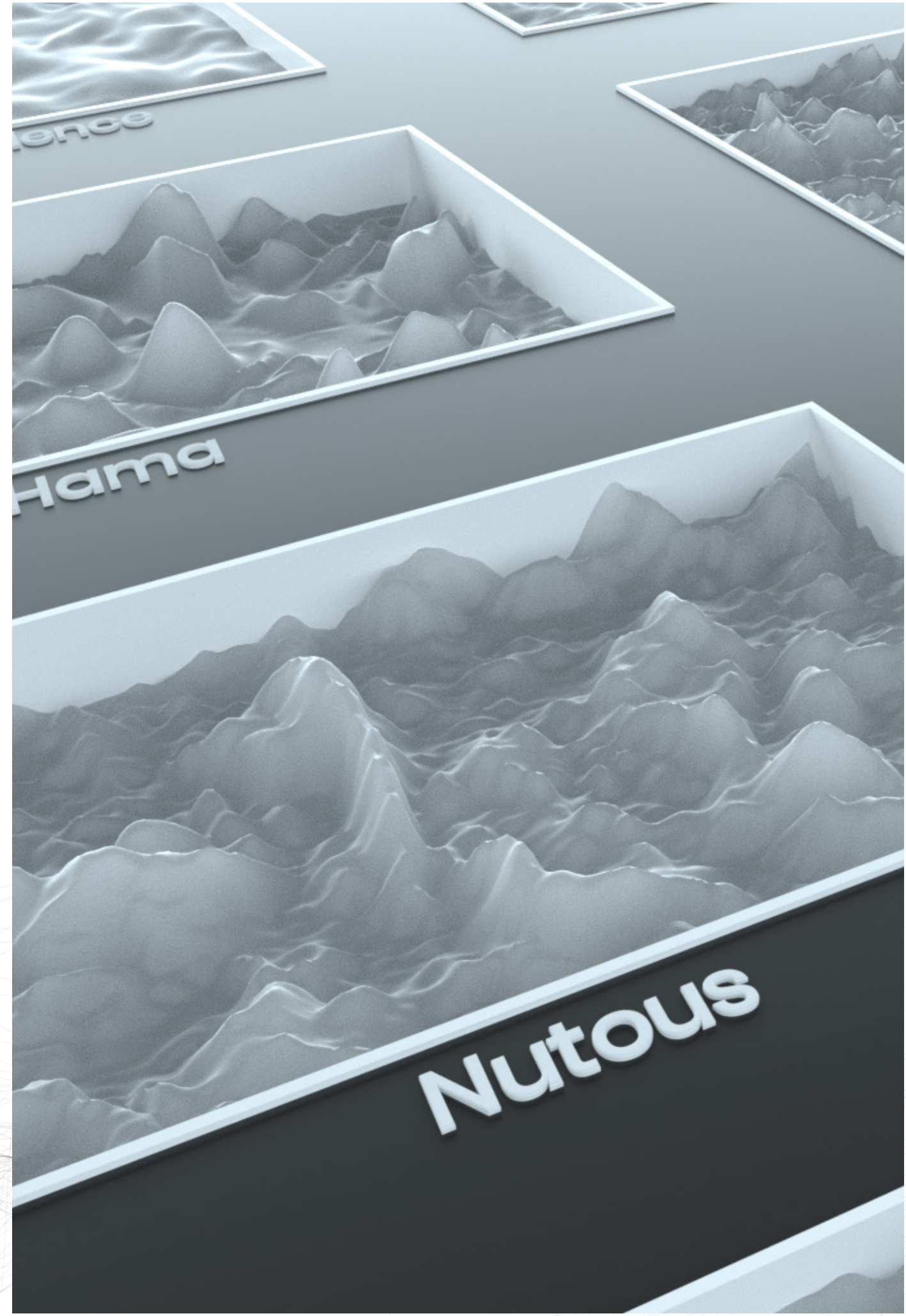

Octane Node Editor
Edit Create View Help
Get Active Mat
Search
Octane Material
Mix Material
Image Texture
Rgb Spectrum
Gaussian Spectrum
Float
W Coordinate
Baking texture
Projection
Transform
Checker
Dirt
Falloff
Marble
Noise
OctDiffuse
Basic Shader
Shader
Texture
Noise
Sampling
Blur Offset
Blur Scale
Amount
Mid level
Level of details
Filter type
Filter radius
Diffuse
Roughness
Bump
Normal
Displacement
Opacity
Transmission
Emission
Medium
Displacem
Input
Create Edit Function Texture

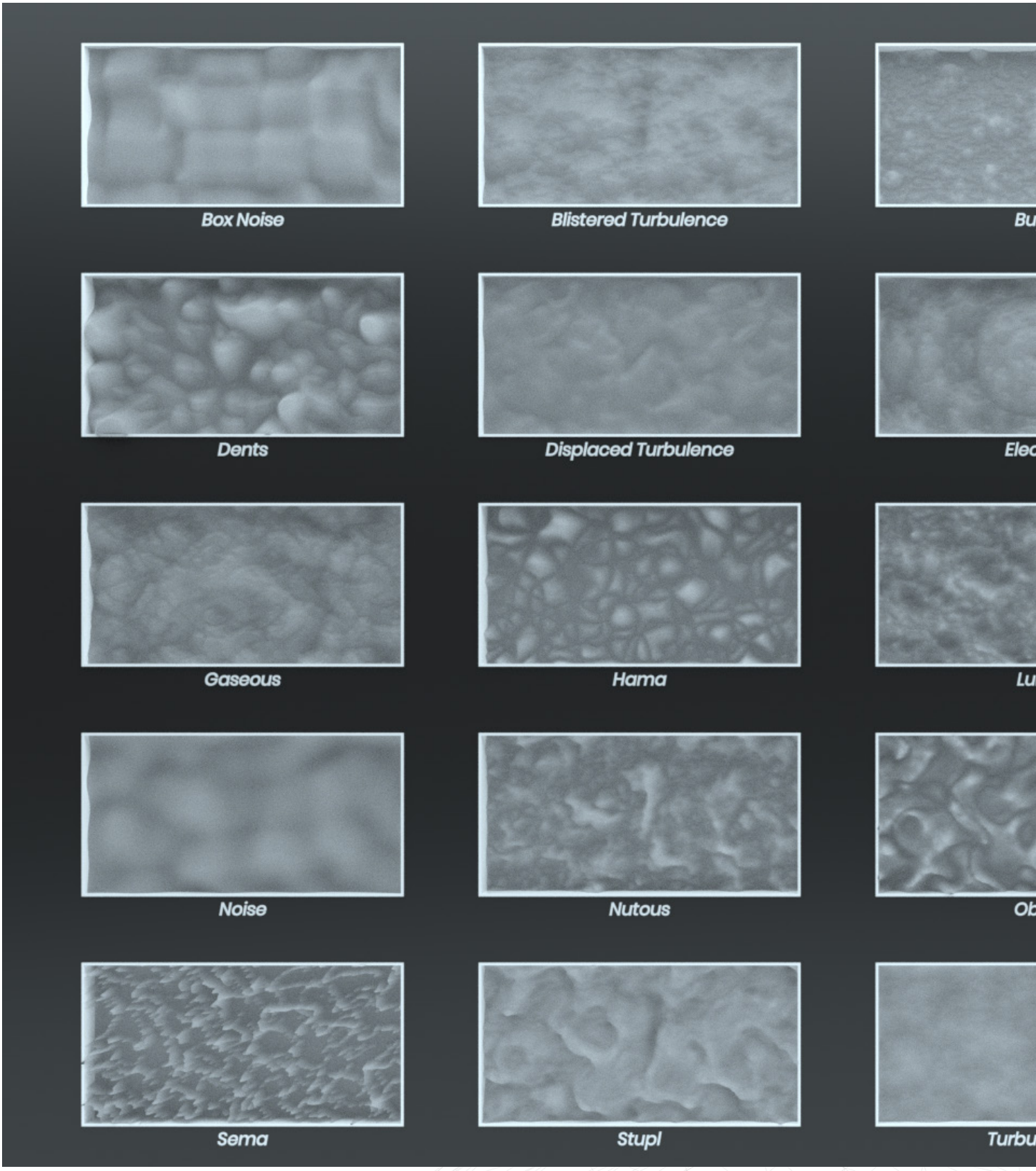

Octane Node Editor
Edit Create View Help
Get Active Mat
Search
Octane Material
Mix Material
Image Texture
Rgb Spectrum
Gaussian Spectrum
Float
W Coordinate
Baking texture
Projection
Transform
Checker
Dirt
Falloff
Marble
OctDiffuse
Basic Shader
Shader
Texture
Noise
Sampling
Blur Offset
Blur Scale
Amount
Mid level
Level of details
Filter type
Filter radius
Diffuse
Roughness
Bump
Normal
Displacement
Opacity
Transmission
Emission
Medium
Displacem
Input
Create Edit Function Texture

D’un autre côté, si la déformation est établie par un « displacer » de Cinema4D, le détail sera défini par les subdivisions que compose la géométrie. Plus cette dernière présente de subdivisions, plus le bruit est affiné et net.





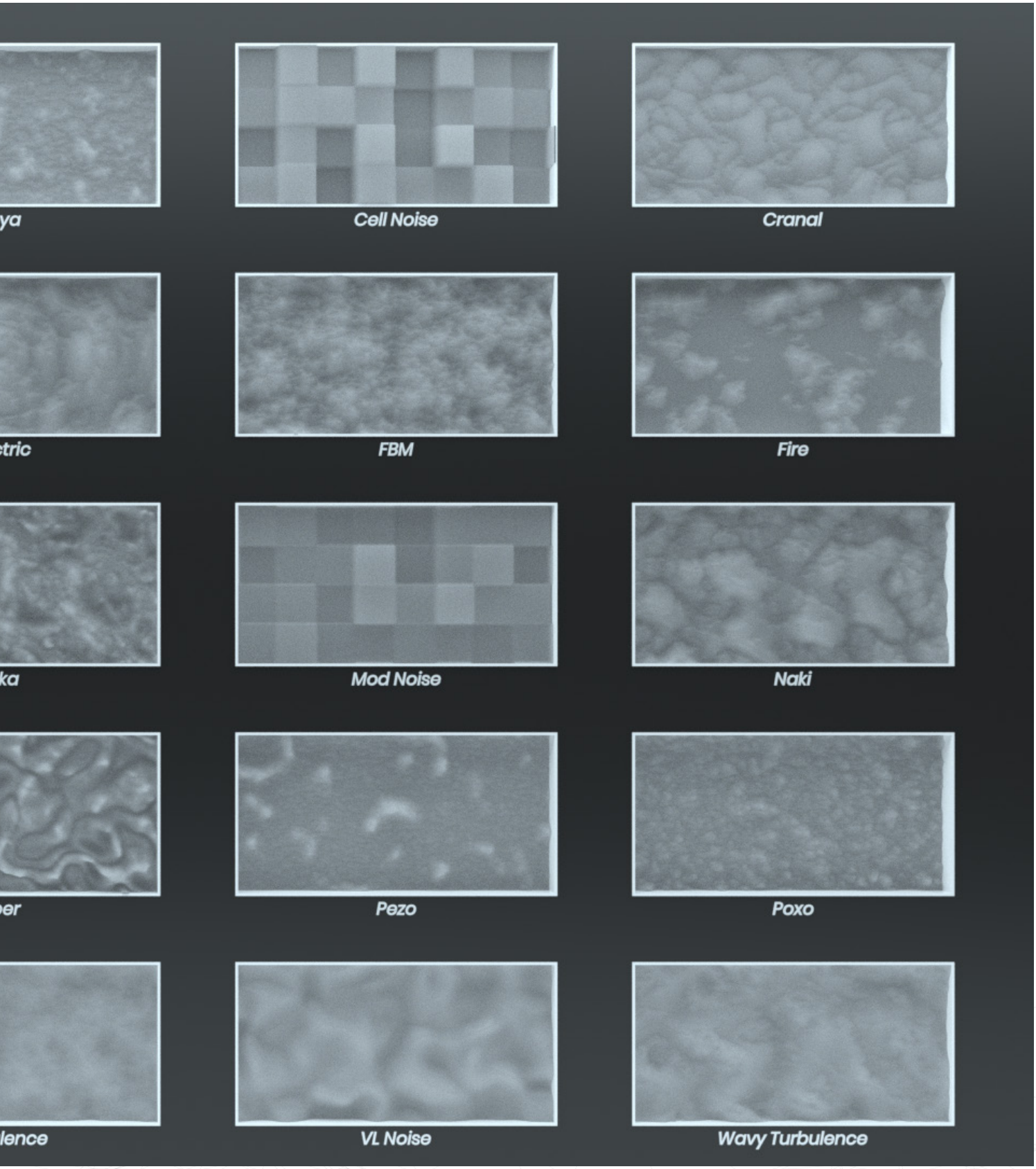

Hama
Nutous



Nuancier

Annexes p.68

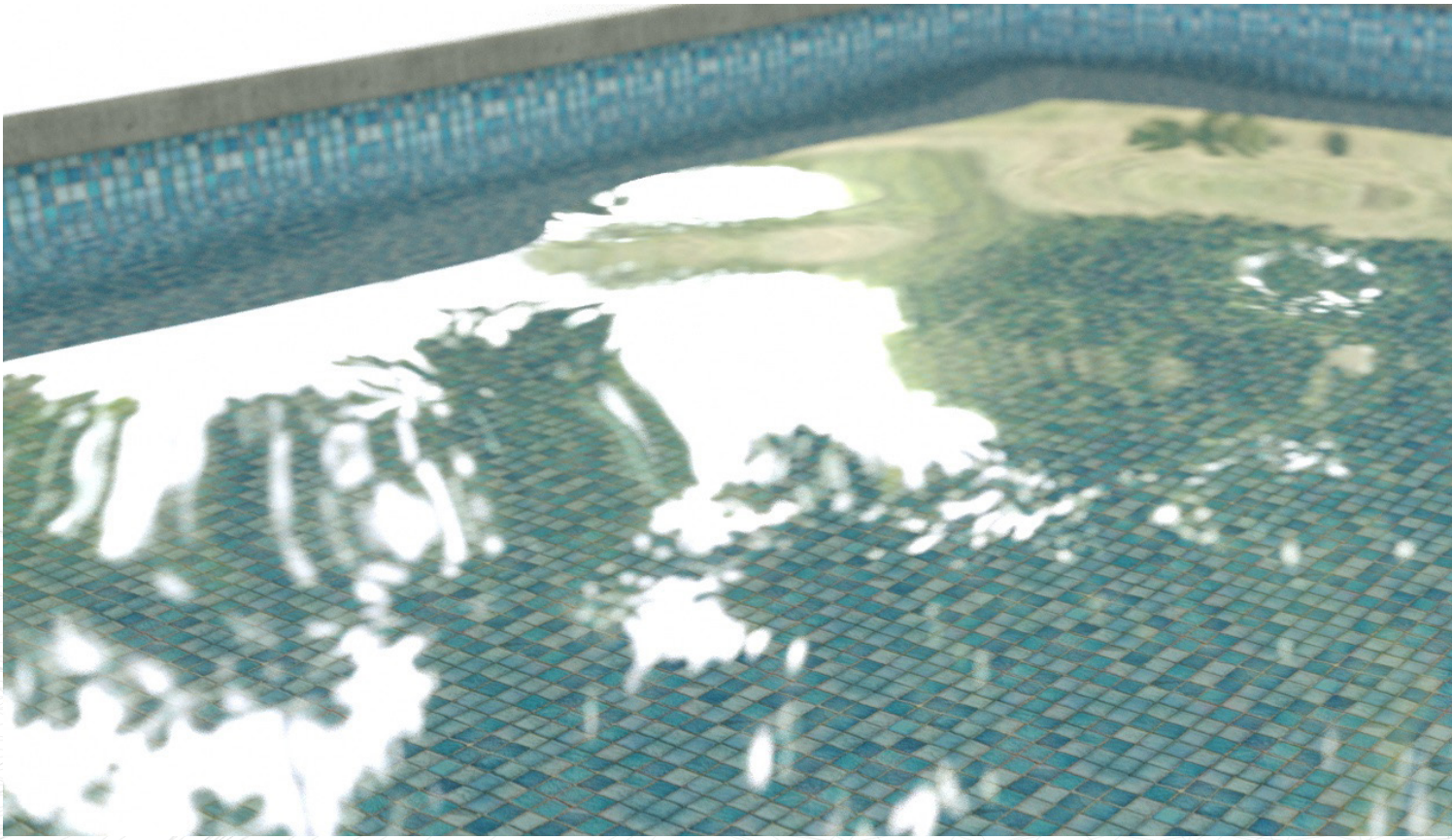
La première chose à entreprendre pour bien visualiser les différents bruits de Cinema4D est de réaliser un nuancier de ceux-ci. C'est une façon concrète de représenter visuellement la palette de bruits qu'offre le logiciel.



Si certains bruits présentent des similitudes, c'est dans leurs mouvements qu'ils se distinguent. En effet, le « Cell Noise » semble ne pas être si différent du « Mod Noise » aux premiers abords. C'est seulement en voyant leurs mouvements générés aléatoirement que la nuance est perçue.

Animation Cell Noise : https://www.ludwigdejonckheere.com/noise/assets/nuancier/cell_noise.mp4

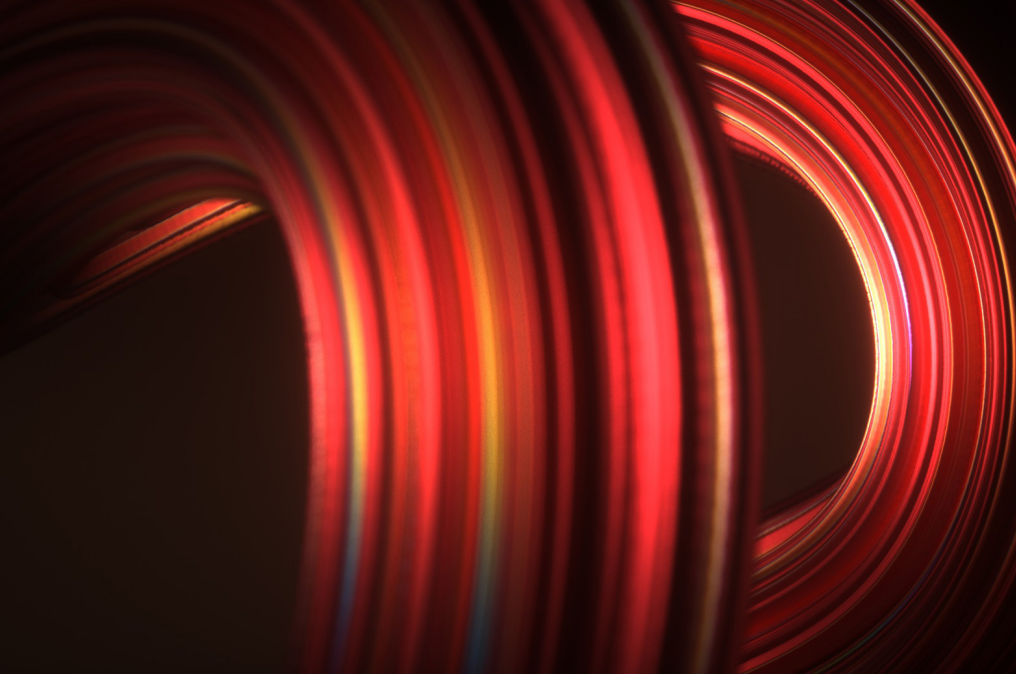
Animation Mod Noise : https://www.ludwigdejonckheere.com/noise/assets/nuancier/mod_noise.mp4



Rendu d’eau

Annexes p.70

Un des usages utiles avec un bruit en tant que déformeur est le rendu de l’eau. En effet, il est très simple et rapide de créer une surface simulant un relief d’étendue d’eau, telle une piscine. Cet usage peut s’avérer intéressant dans le rendu architectural ou encore d’environnements naturels. De plus, cette méthode fonctionne aussi s’il s’agit d’une animation, en modifiant alors le paramètre de vitesse du bruit.



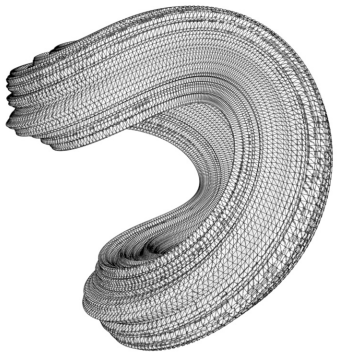
Échelle Relative

Annexes p.71

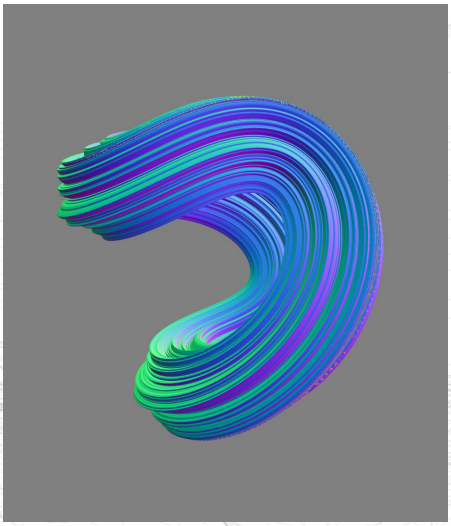
Lorsque le bruit est agrandi ou rétréci de manière relative selon les axes, il apparait alors distendu et écrasé. Cet aspect rend alors le bruit comme étant allongé et présente des lignes droites. Ceci permet alors de créer des sillons le long d'un objet 3D.

Dans cet exemple, il s'agit d'un torus, une sorte d'anneau, où un « bend » lui est appliqué premièrement. Le bruit distendu lui est ensuite appliqué pour lui donner cet aspect irrégulier sur sa longueur. Pour obtenir cet aspect, l'échelle relative à l'axe Y a été augmentée de 10 000%.

Occlusion ambiante



Wireframe



Normals

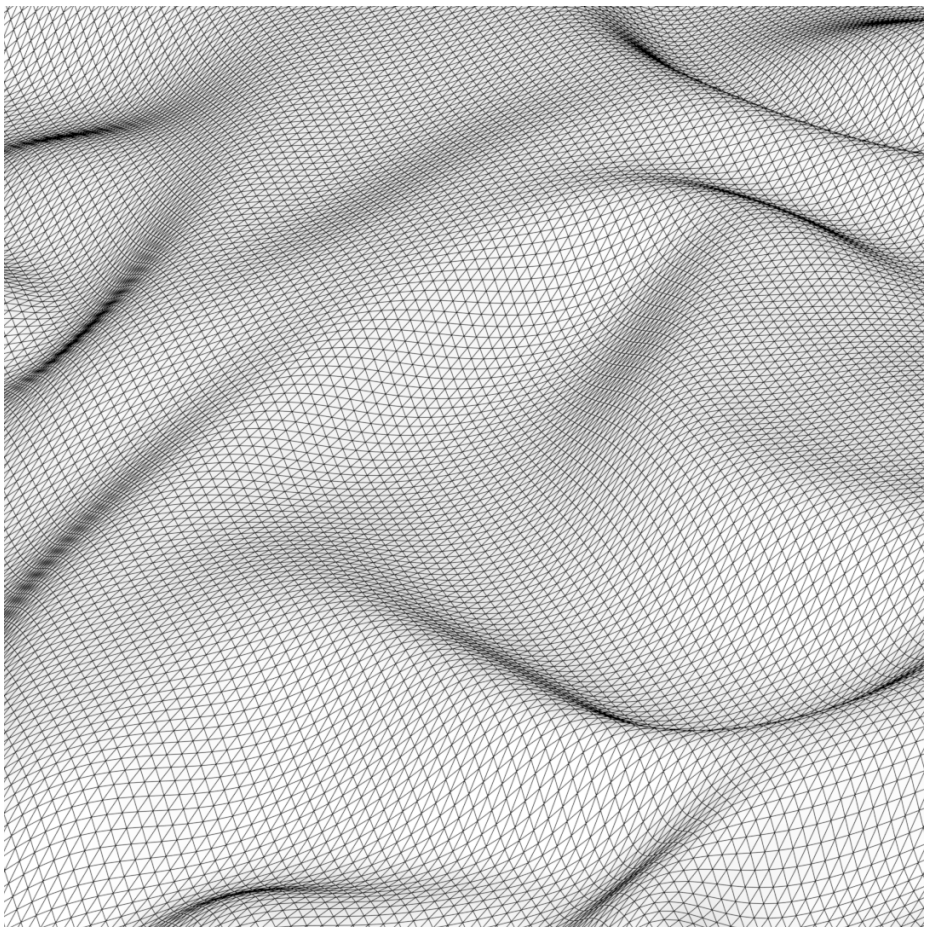
Iridescence

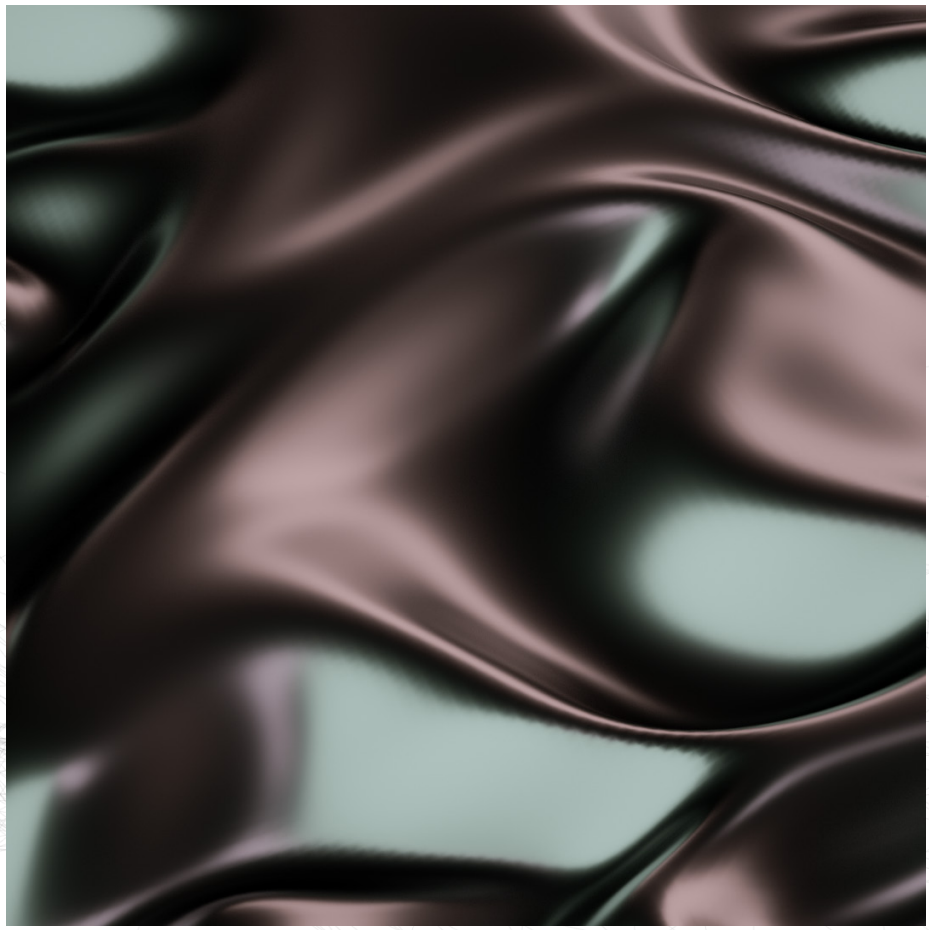
Annexes p.72

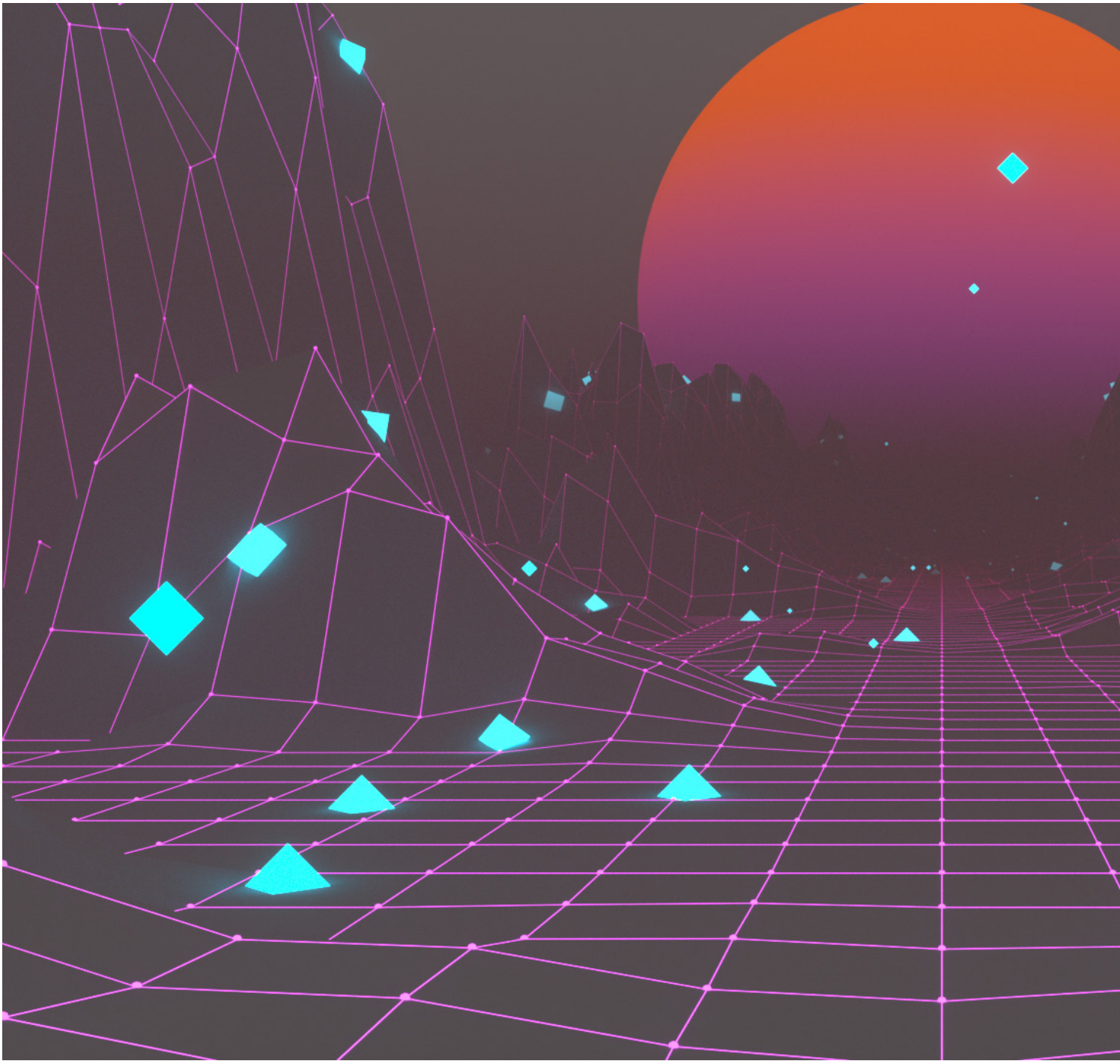
Une autre application, similaire à celle du rendu d'eau, est d'utiliser le bruit en displacer de manière animée sur une surface plane. En lui attribuant un matériau réfléchissant à la manière du métal, la surface va réfléchir les lumières de différentes couleurs en prenant un aspect bariolé.

Pour permettre une géométrie fluide sans artéfacts, la surface plane contient un nombre important de subdivisions.

Animation Iridescence : https://www.ludwigde-jonckheere.com/noise/assets/iridescence/iridescent.mp4



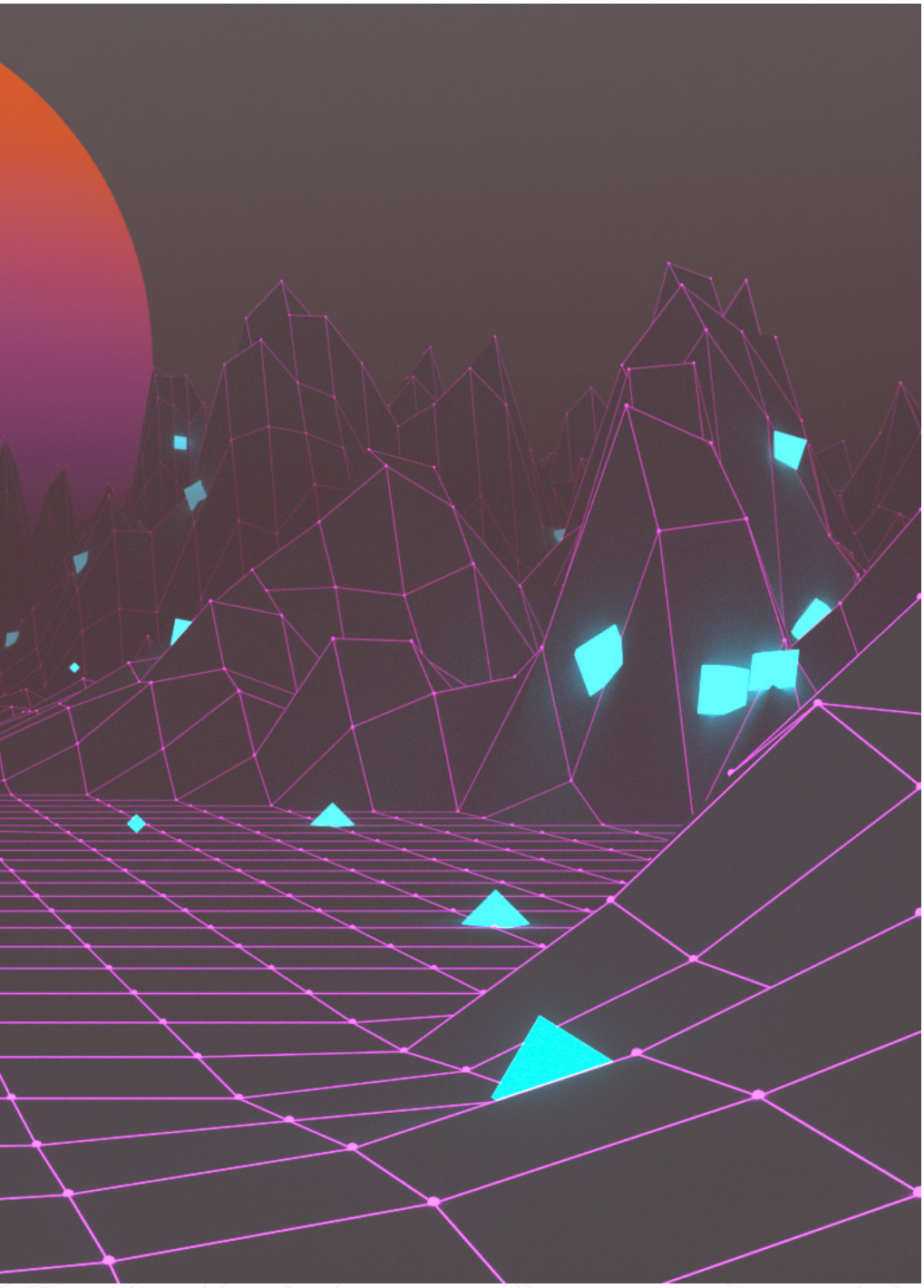




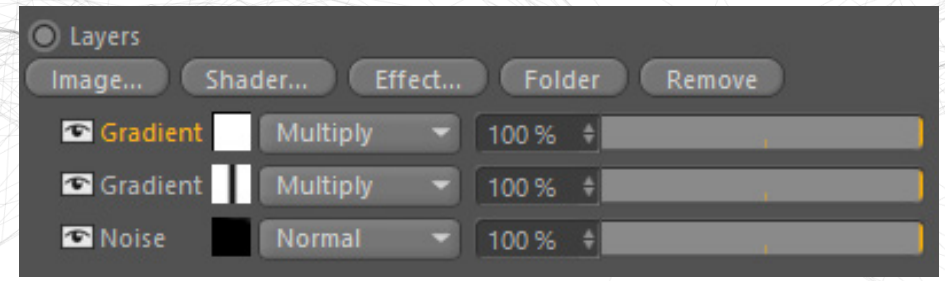
Rétro Terrain

Annexes p.72

Le choix d'un faible nombre de subdivisions dans la géométrie est aussi un choix esthétique qui peut dans cet exemple reproduire ce look « Vaporwave ». Lorsque la surface contient peu de subdivisions, le bruit est directement moins détaillé et donne alors ce résultat de bosses anguleuses.



Afin de ne pas rendre le displacer effectif au centre de la composition et ainsi créer un « chemin » rectiligne, le bruit est ici appliqué en tant que calque, sur lequel s'ajoute un dégradé noir au milieu et blanc sur les côtés, en mode de fusion « multiply ». De cette manière, le milieu est toujours de couleur noire et le displacer n'affecte donc pas la géométrie.



Animation Retro Terrain : https://www.ludwigde-jonckheere.com/noise/assets/retro.mp4

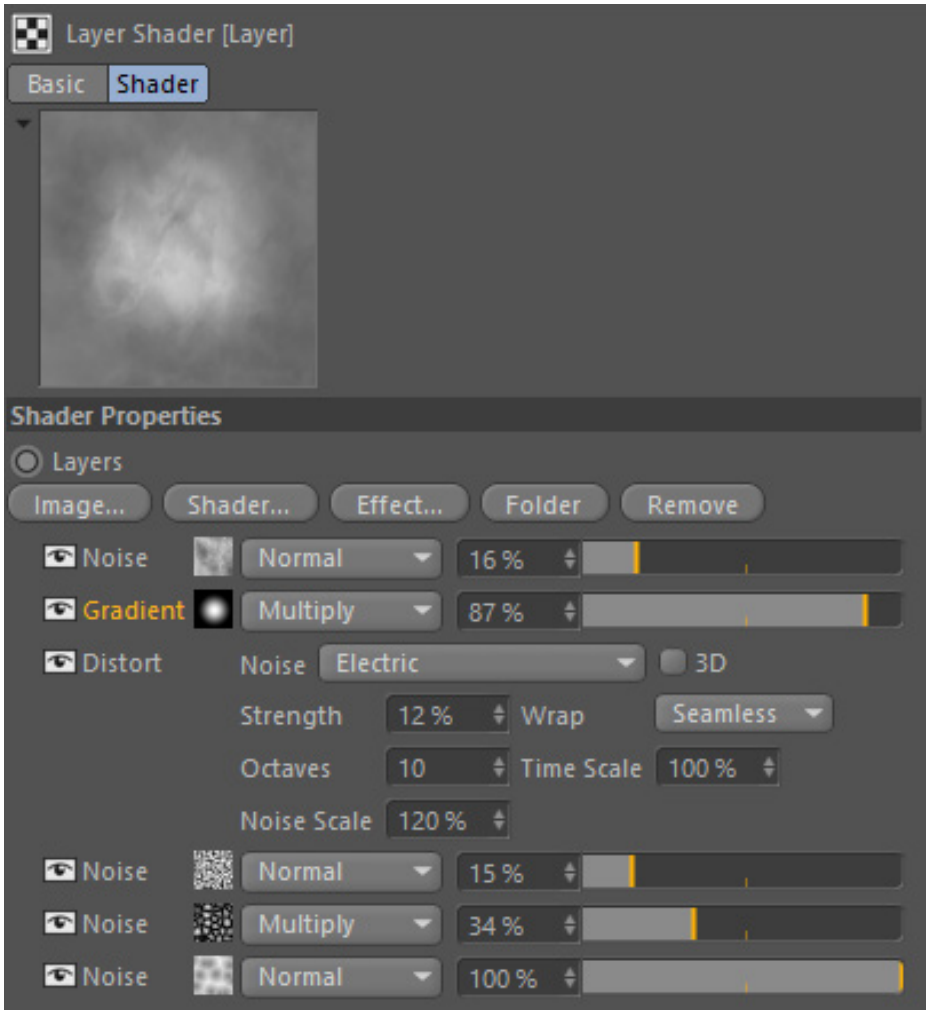
Montagne

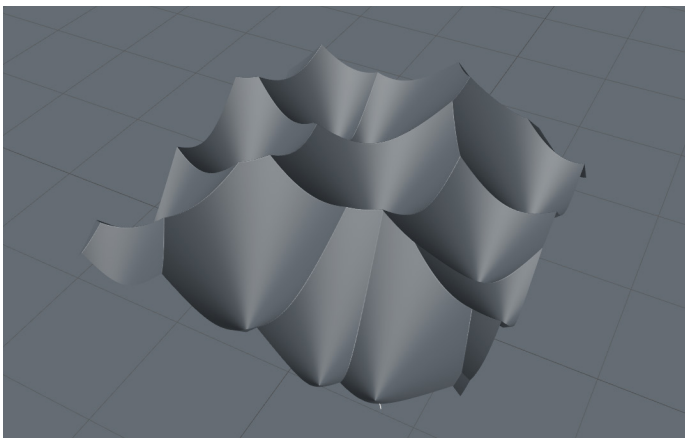
Annexes p.73

Pour ce dernier exemple d'application de bruit dans le « displacement », l'objectif est de créer une chaine de montagnes. Il existe un outil de création de terrain au sein de Cinema4D mais le but ici est de créer ces montagnes avec le displacement.

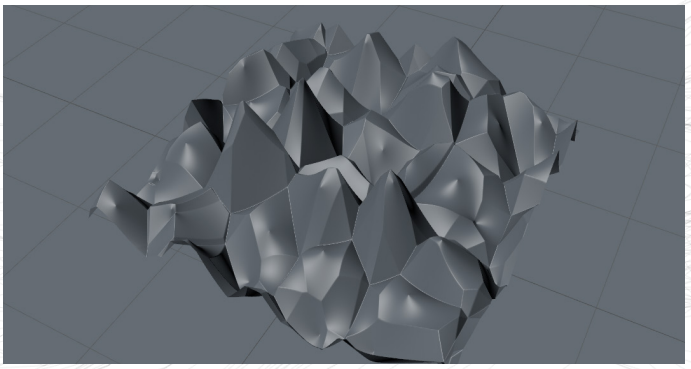
Ceci consiste donc en une multitude de couches dans le displacer qui agiront plus ou moins sur la géométrie.

Comme ces calques fonctionnent à la manière de ceux d'Adobe Photoshop, il faut commencer la lecture par le bas.





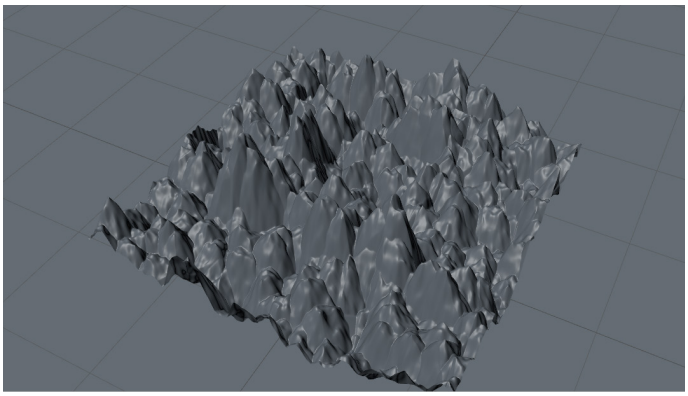
Dans un premier temps, il faut générer une géométrie de « gros œuvres » qui pose le volume global de la montagne. C'est pourquoi l'échelle de ce bruit est agrandie par rapport aux autres (ici 400%).



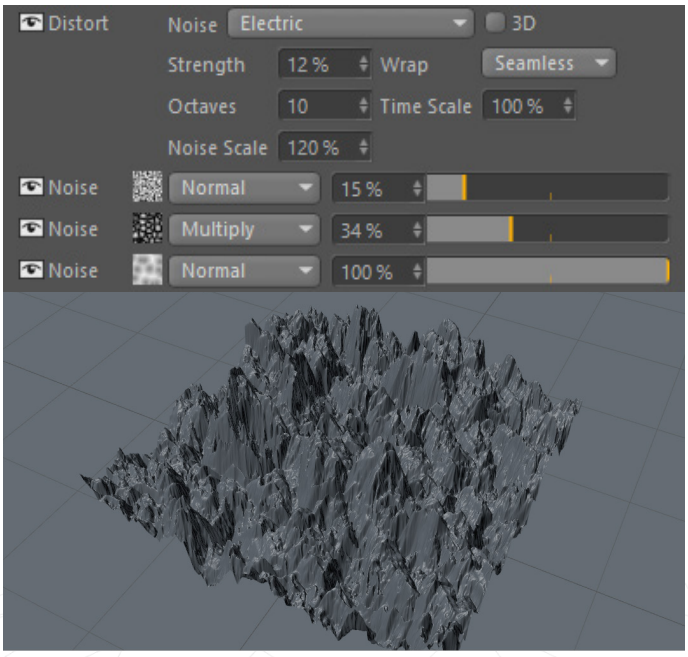
L'étape suivante est d'ajouter une couche de détails moyens qui garde la forme du premier volume et qui intègre un relief un peu plus détaillé. L'échelle de ce bruit est légèrement plus petite pour permettre de ne pas rentrer directement dans le détail (250%).

S'ajoute à ceci le fait que ce calque est en opacité moindre (ici 35%), en mode de fusion « multiply ».

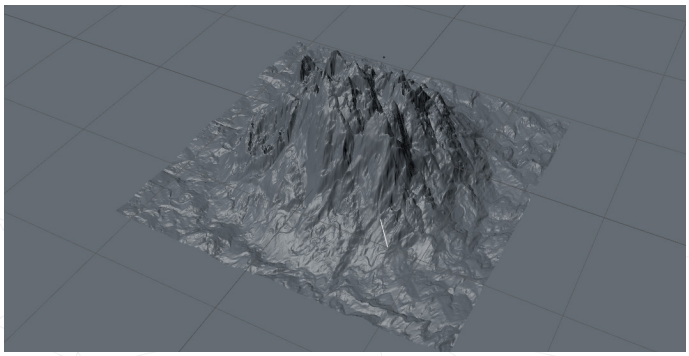
La troisième couche, quant à elle, génère du détail plus fin et présente déjà une forme plus organique. Cette couche est laissée en mode normal, avec une faible opacité (15%).



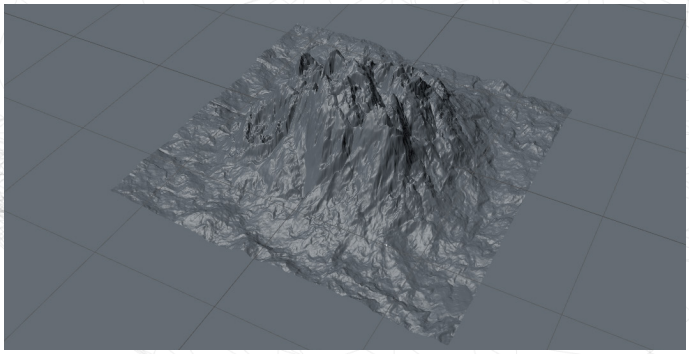
Afin de venir maintenant creuser dans le détail, un effet est rajouté en mode « multiply ». Cet effet est une distorsion selon un bruit. Cette couche rajoute de la complexité et de l'aléatoire à la géométrie.



Enfin, pour donner à voir la topologie d'une montagne et non pas d'un terrain vague, un dégradé est rajouté sur le tout, afin que le displacer ne prenne pas effet sur les bords.



La forme de la montagne est quasiment terminée, un dernier bruit est rajouté en texture fine, en mode normal, à 16% d'opacité.



TEXTURING

Qu’est-ce que le texturing ?

Le « texturing » se réfère à la création de matériaux. Un matériau est constitué de plusieurs caractéristiques permettant de recréer quasiment tous les matériaux existant dans le réel, qu’il s’agisse de verre, de pierre, de plastique, etc.

Dans ce contexte, le texturing se fait via l’éditeur de matériaux d’Octane, le moteur de rendu et non via l’éditeur de matériaux standards de Cinema4D.

Parmi ces paramètres, il y a :

- « **diffuse** » : la couleur générale du matériau, que ce soit une couleur unie, un dégradé, un bruit, etc.

- « **roughness** » : la rugosité du matériau. Une faible rugosité rend la surface brillante, tandis qu’une forte rugosité la rend plus mat.

- « **bump** » et « normal » : ces deux paramètres simulent un relief depuis une texture. L’apparence du matériau change et peut donner cette illusion alors que la géométrie est inchangée.

- « **displacement** » : paramètre relatif au chapitre précédent, mais il s’agit ici du matériau, et non pas de l’objet 3D. Cependant, contrairement au « bump » et « normal », la géométrie est modifiée.

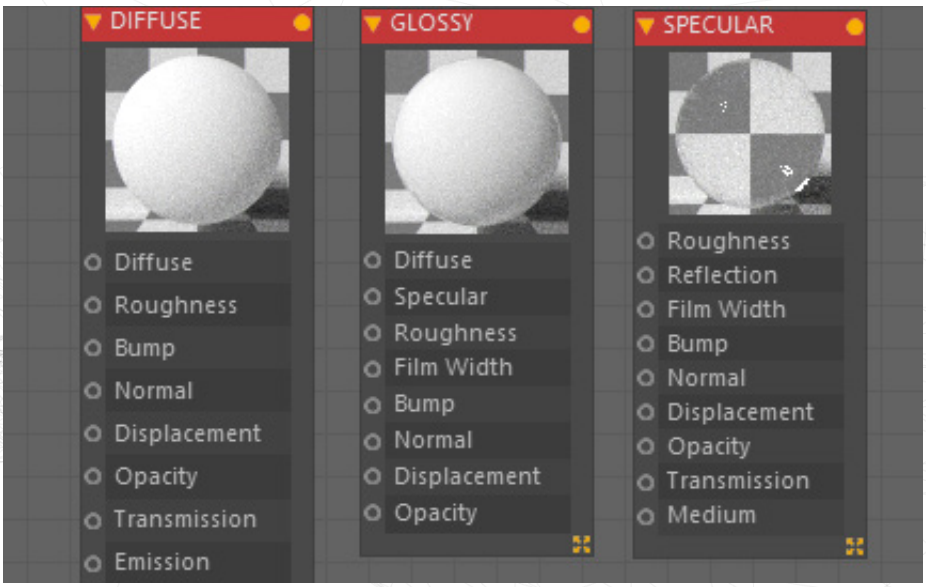
- « **opacity** » : tout simplement l’opacité du matériau.

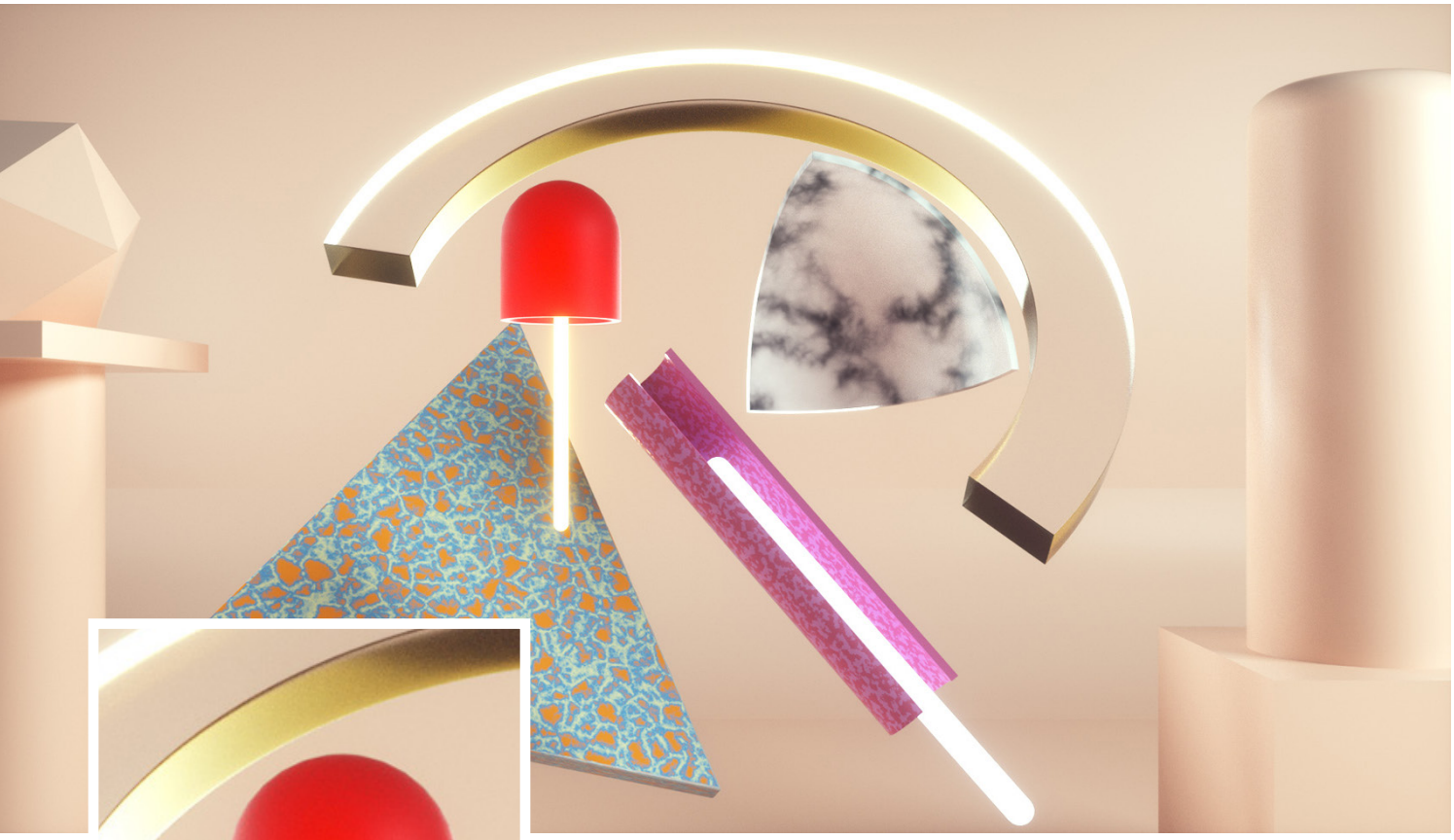
- « **emission** » : le matériau devient une source lumineuse.

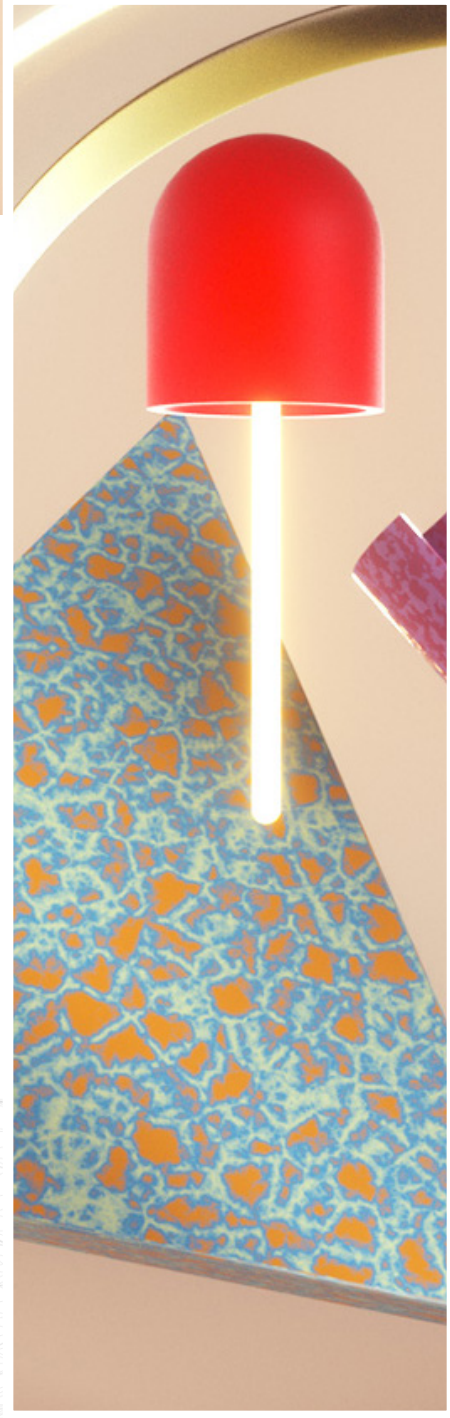
Il y a bien d’autres paramètres qui varient selon le matériau, mais ceux-ci sont les plus importants pour la suite.

Chaque paramètre décrit ci-dessus peut se voir attribuer une texture et, en l’occurrence, un bruit.

En ce qui concerne le bruit en lui-même, il est question, dans la plupart des cas, du bruit d’Octane et non de Cinema4D. C’est presque similaire, mais les paramètres sont présentés différemment.







Diffuse

Annexes p.74

En commençant par la caractéristique « diffuse » du matériau, le bruit agit directement sur la couleur de la texture. Le bruit est basiquement une nuance de noir et blanc, mais en lui associant un dégradé d'Octane, le bruit présente alors des variantes de couleurs. Ce dégradé contient au minimum deux couleurs, mais peut aussi en présenter davantage.

Roughness

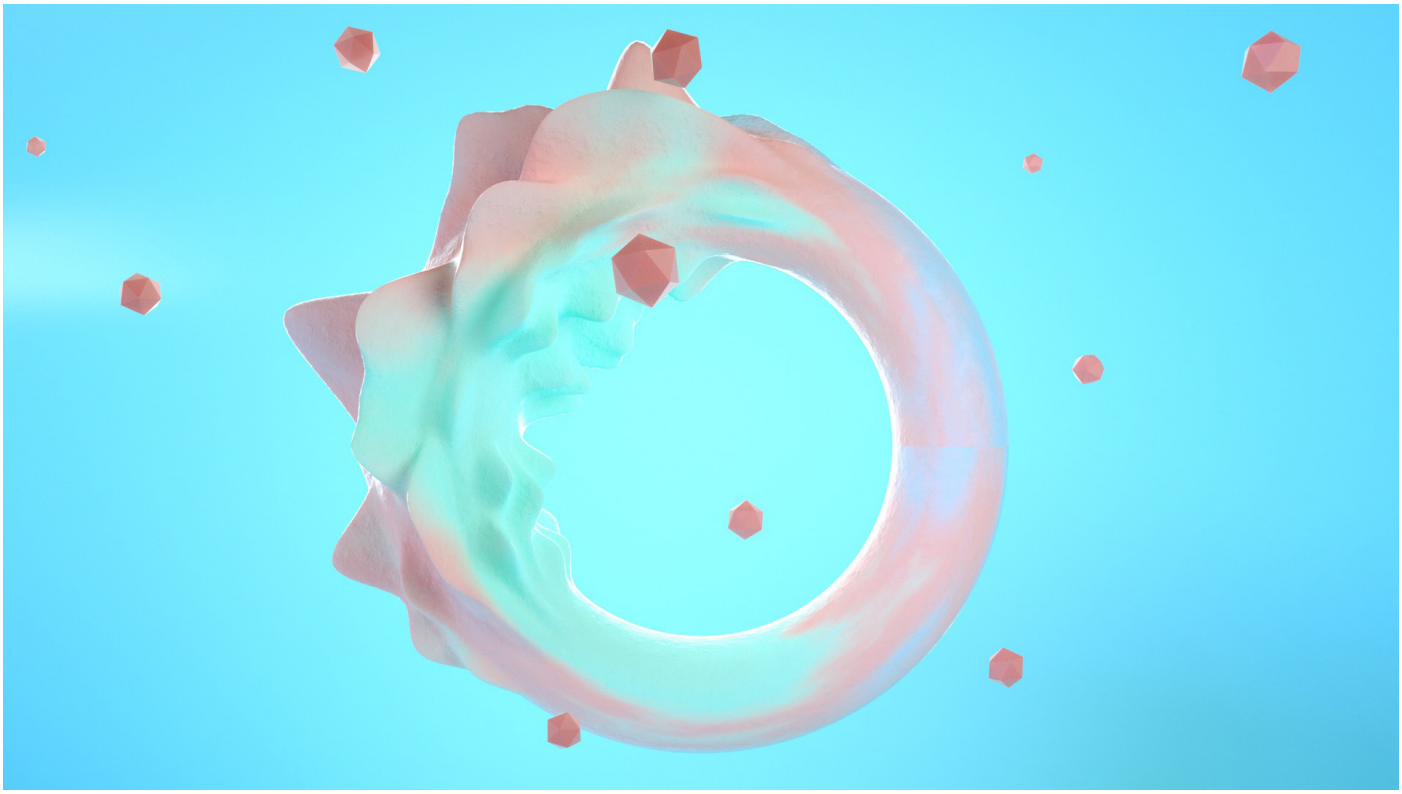
Annexes p.74

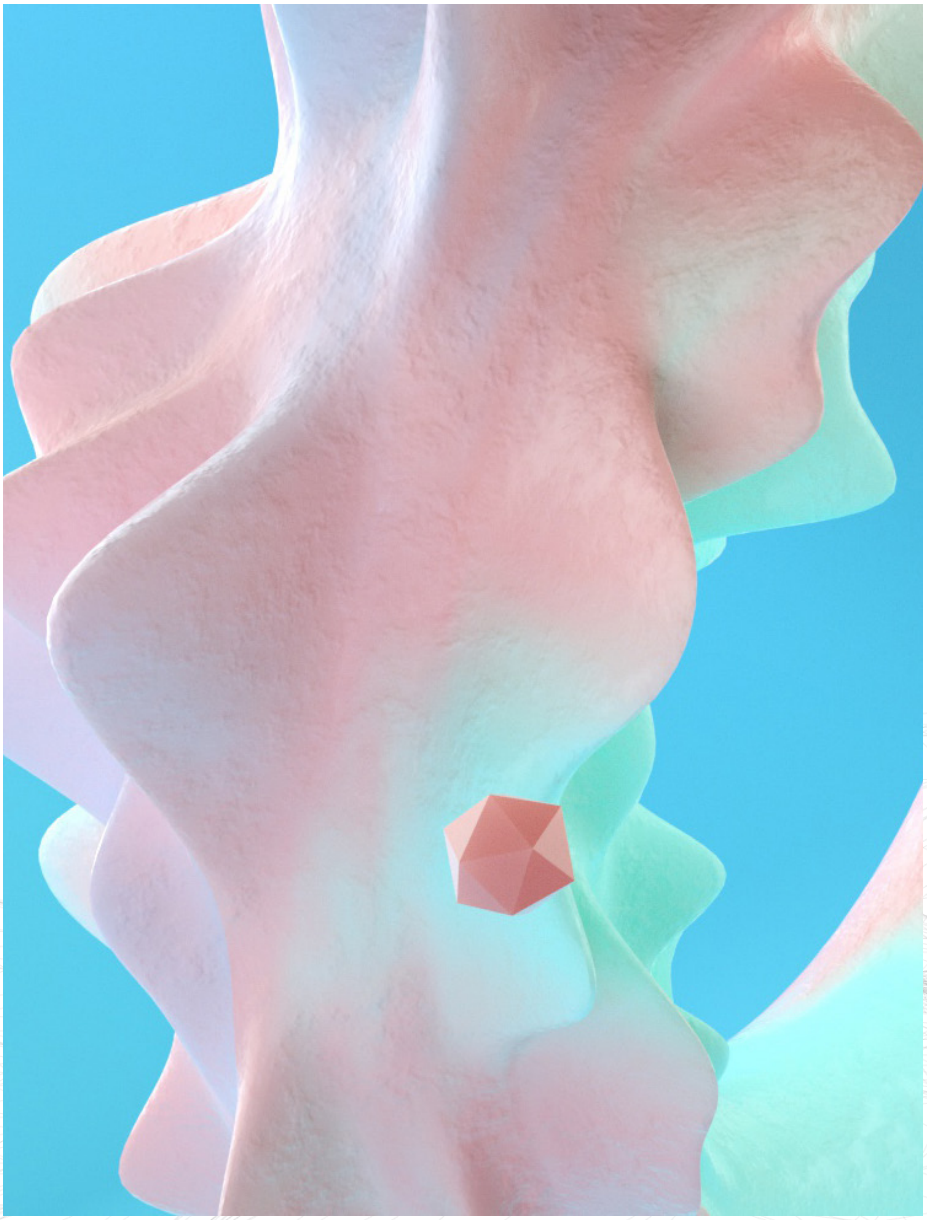
La rugosité d'un matériau va drastiquement changer son comportement vis-à-vis des reflets de lumière. Plus le matériau est lisse, plus les reflets seront définis et nets. Si celui-ci est rugueux, les reflets seront alors plus atténués et doux.

Ce paramètre peut varier avec une valeur allant de 0 à 1, 0 étant la valeur la plus lisse et 1 la valeur la plus rugueuse. Mais la rugosité peut aussi être définie en fonction d'une texture, et donc un bruit. Les nuances sombres et noires de ce bruit sont associées à l'aspect lisse et les nuances claires et blanches sont alors rugueuses.

Une bonne application du bruit dans le paramètre de rugosité est frappante lorsqu'il s'agit de matériaux spéculaires, comme du verre par exemple.

Dans cet exemple, il est question d'une forme géométrique, l'icosaèdre, auquel un matériau similaire à du verre est appliqué. Au niveau de la rugosité est attaché une texture de bruit, qui donne alors un aspect gelé ou sali au verre.





Bump

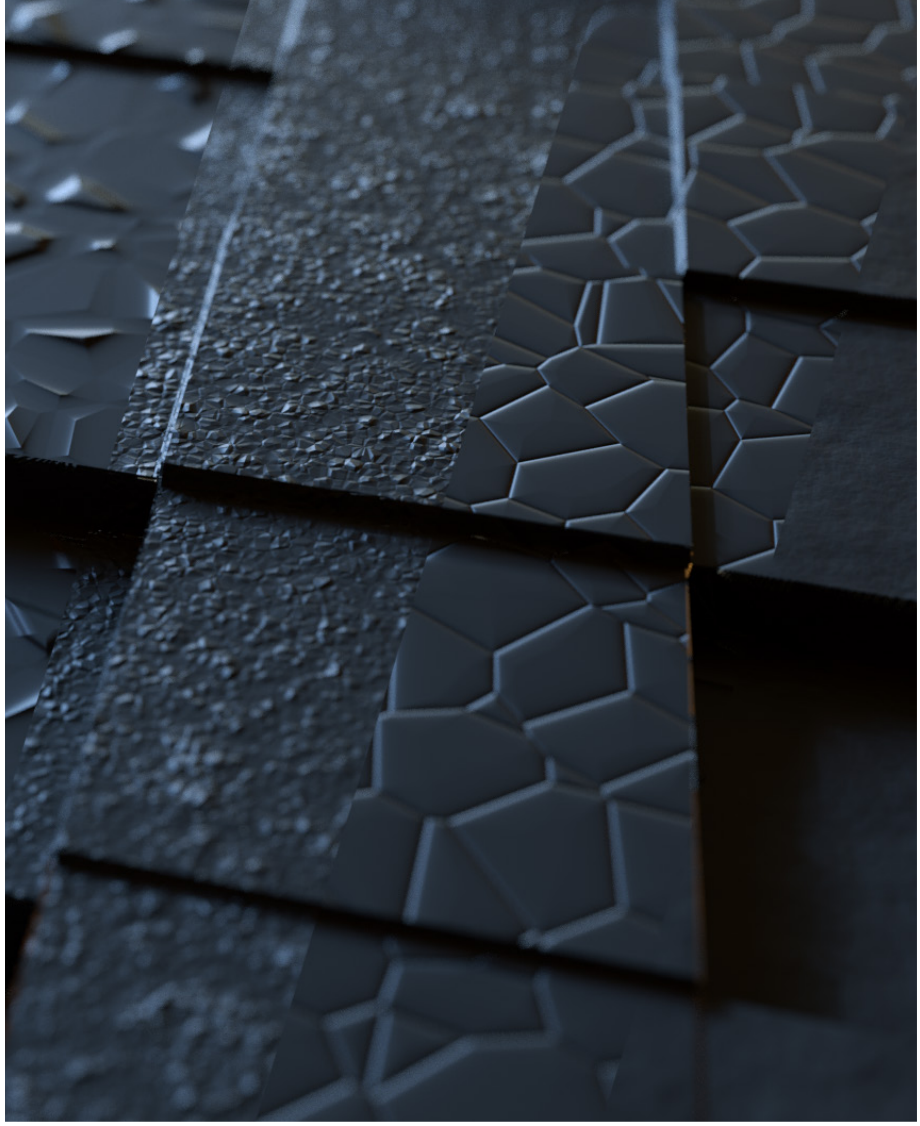
Annexes p.75

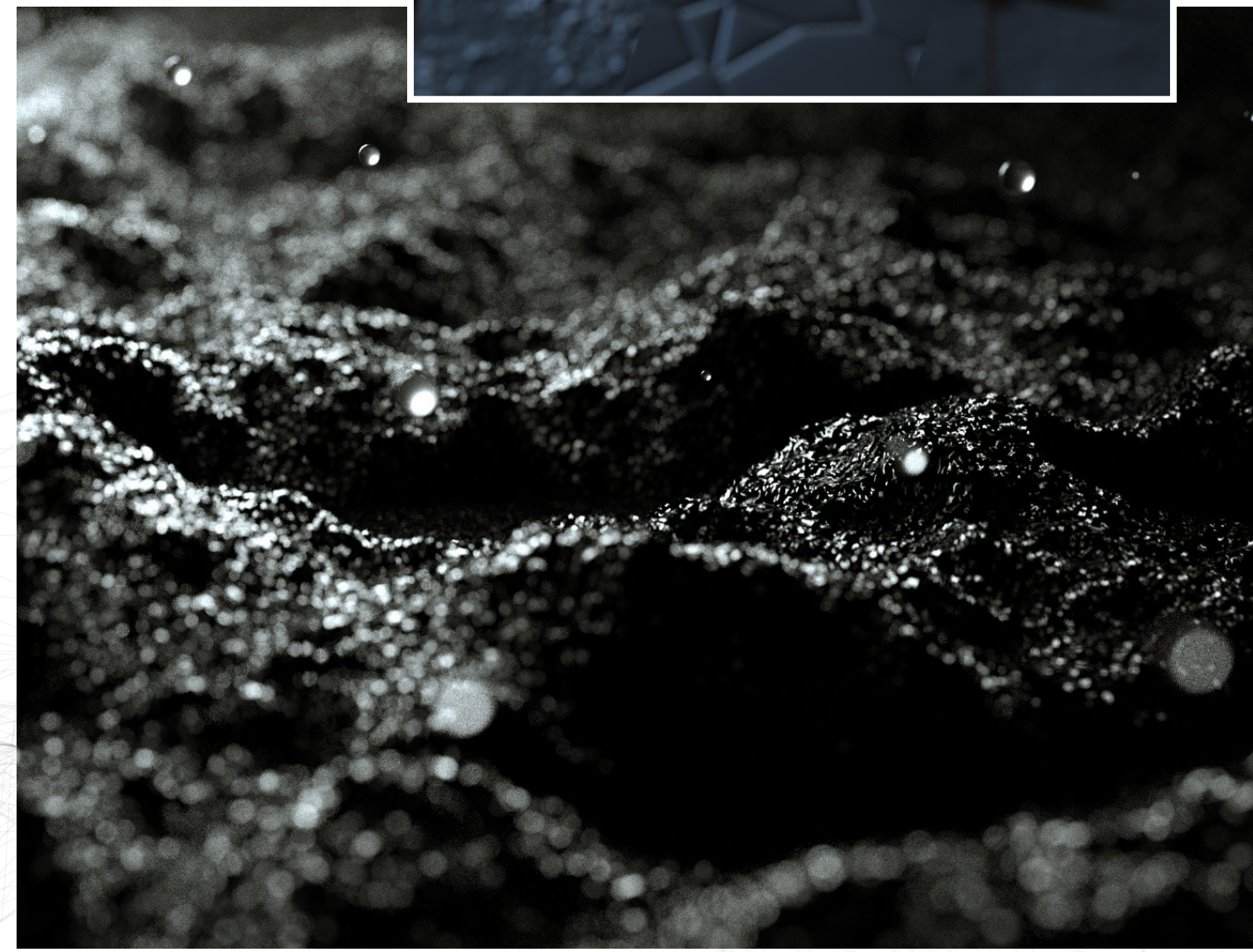
Le paramètre de « bump » consiste à simuler un relief, au même titre que les « normals ». Cependant, le paramètre de normals prend ici en compte un autre type de texture contenant des variances de couleurs selon les axes X, Y et Z. Un bruit est donc difficilement applicable. Mais ce n'est pas le cas du « bump », qui s'adapte aux nuances noires et blanches.

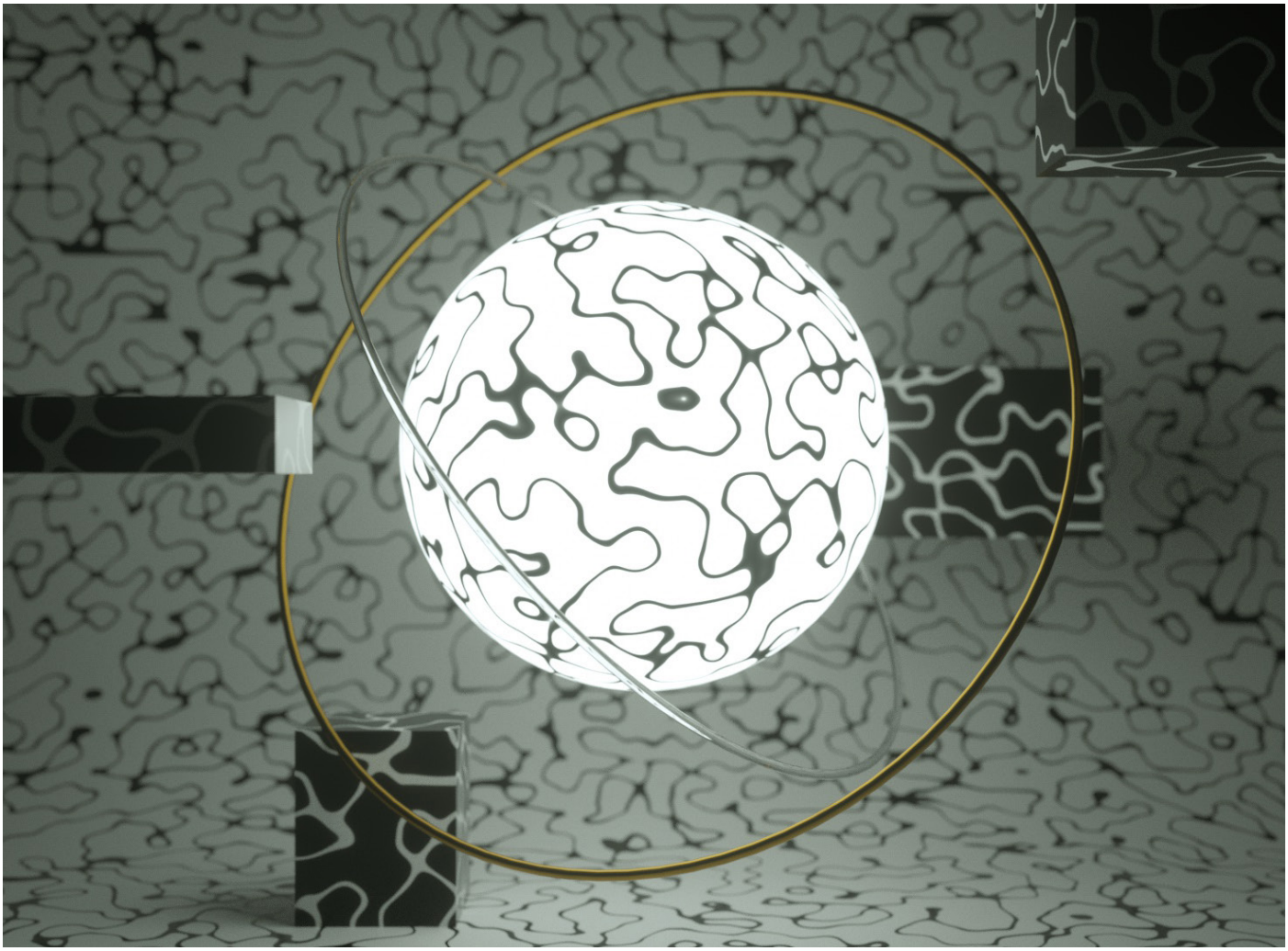
Ce paramètre est assez similaire au « displacement », à la différence que la géométrie ici reste inchangée, le relief n'agit pas véritablement sur la topologie.

Dans cet exemple de torus, le détail de la texture est assez fin et est généré via un bruit dans le « bump ». Ceci rend alors le matériau plus intéressant et riche.

Il existe également une possibilité de rajouter du détail supplémentaire à plus petite échelle sur des surfaces qui présentent déjà un « displacement ». En mixant alors ces deux paramètres, on obtient un terrain complexe et organique.







Emission

Annexes p.78

Enfin, le dernier paramètre de texture présenté est le canal « emission » ou émission en français. Il s'agit plus vulgairement de rendre une surface lumineuse. Comme pour les paramètres présentés précédemment, un bruit peut très bien s'insérer en tant que source de lumière.

Dans cet exemple, c'est le bruit cranal qui est source de lumière, où la partie blanche devient alors émettrice.

Animation Emission :
https://www.ludwigdejonckheere.com/noise/assets/emission.mp4

VOLUME

Que sont le Volume Builder et le Volume Mesher ?

Ces deux fonctionnalités sont apparues dans la version R20 de Cinema4D et fonctionnent ensemble.

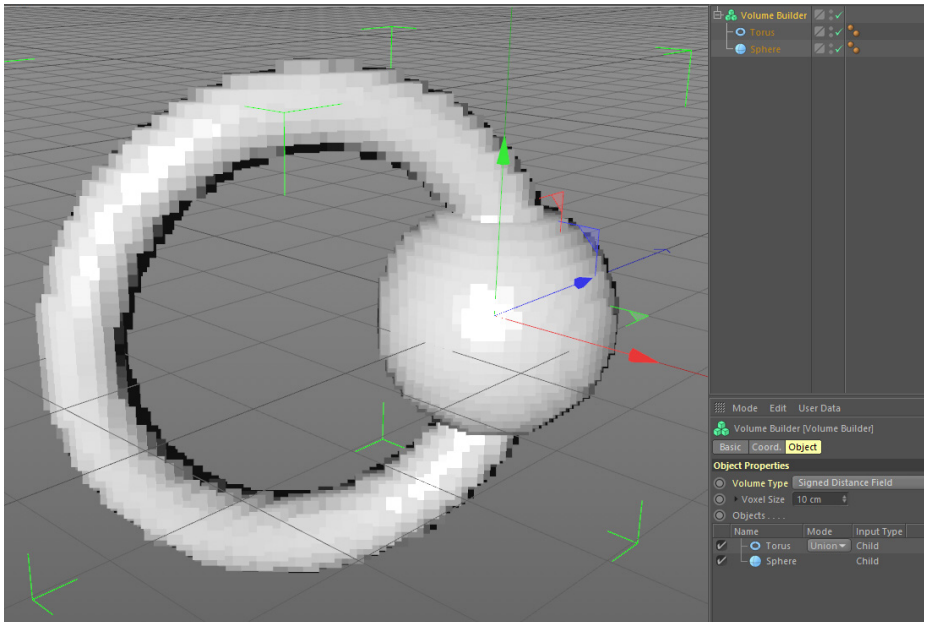
Premièrement est appliqué le « volume builder », sur un/des objet(s) 3D. Le « volume builder » transforme alors la géométrie en un ensemble de « voxels », une sorte de pixels 3D. Ces derniers prennent l'apparence de petits cubes blancs. La taille des pixels 3D peut varier, rendant ainsi le résultat final plus ou moins détaillé.

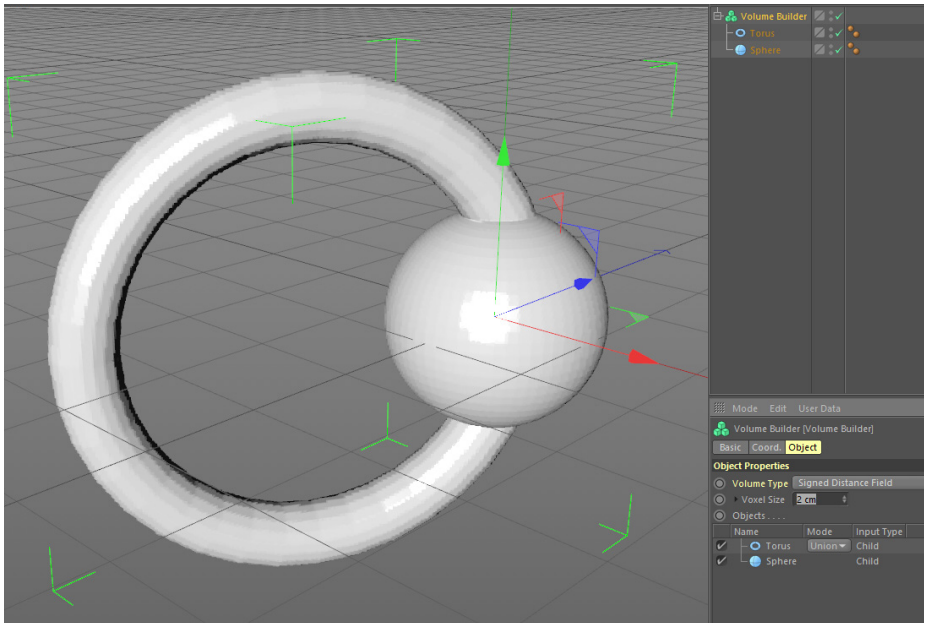
Si la taille des petits cubes est petite, le résultat est plus précis. À l'inverse si les « voxels » sont de plus grande taille, le détail est perdu.

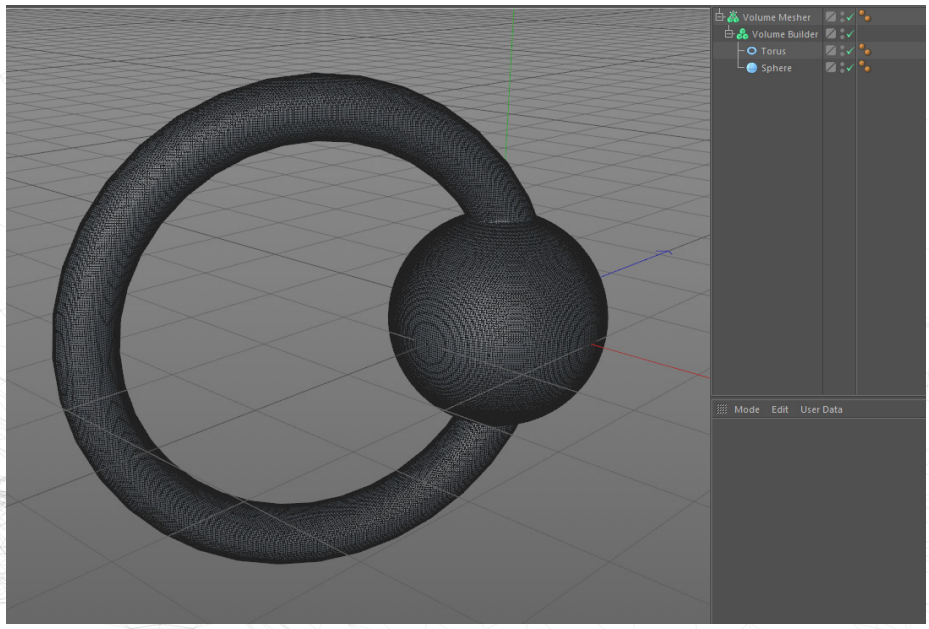
Attention cependant, plus les pixels 3D sont petits, plus le logiciel demande de ressources, et donc ralenti, voire se ferme ou ne répond plus.

Ce « volume builder » va donc transformer une forme géométrique en un ensemble de pixels 3D. Sur un objet 3D simple comme une sphère ou un cube, l'utilité est nulle. Si plusieurs objets complexes sont chevauchés et entremêlés, alors cette fonctionnalité devient intéressante.

En effet, les diverses formes géométriques vont fusionner et se souder les unes aux autres, pour créer une forme globale.



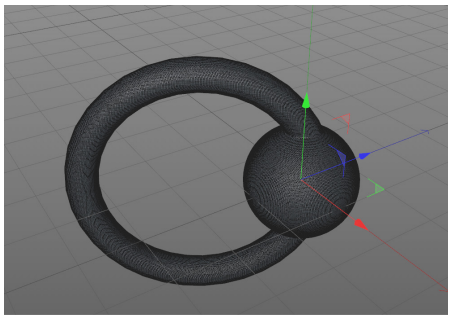




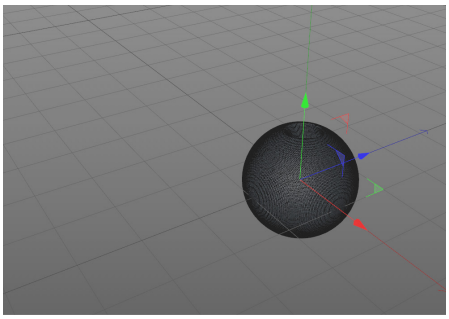
À cette étape, la forme n'apparait encore que comme un ensemble de pixels 3D agglutinés. C'est alors qu'intervient le « volume mesher » qui va retransformer ces cubes en une vraie forme géométrique composée de polygones.

Plusieurs modes sont proposés avec ces nouvelles fonctionnalités, un peu à l'image de l'outils « booléen », un outil très connu dans le monde de la 3D.

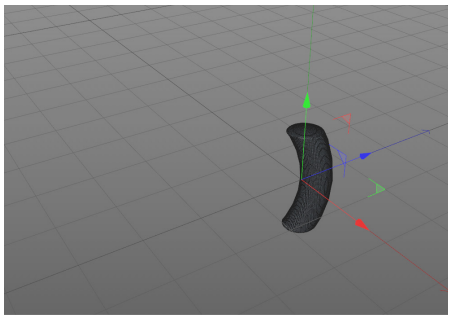
Basiquement, ces modes sont au nombre de 3 :



Union : permet d'unir plusieurs objets 3D pour n'en former qu'un.

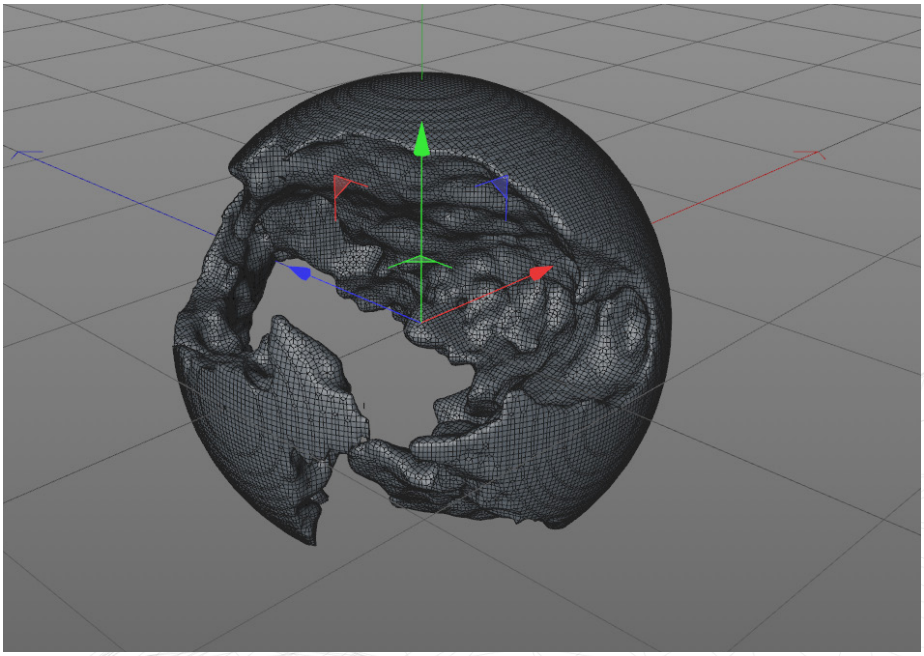


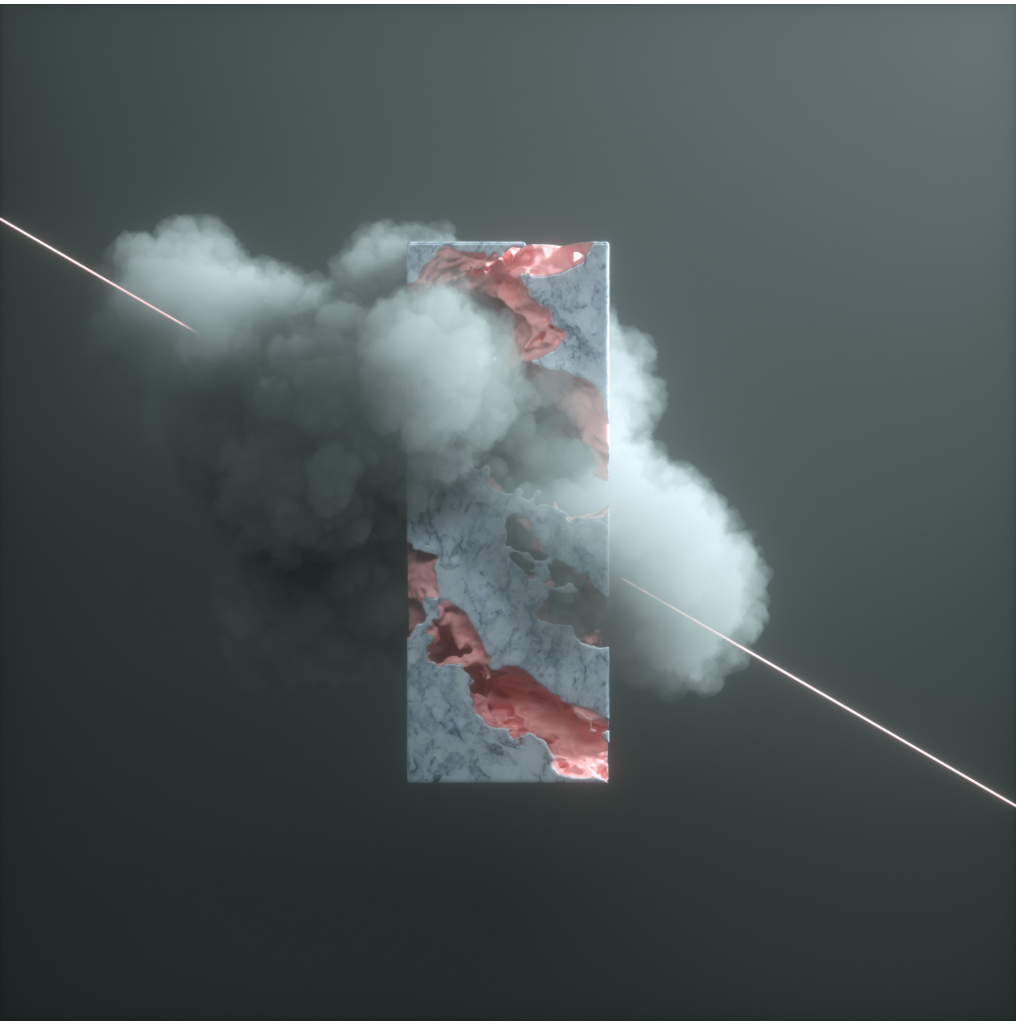
Soustraction : creuse la forme d'un objet en fonction de la forme des autres objets. L'ordre des objets est ici prise en comtpe.



Intersection : ne prend uniquement en compte que la partie intersectée entre les différents objets 3D.

Dans le cadre du bruit, et concernant le « volume builder » et « volume mesher », l'idée est de creuser un objet 3D grâce à un champ aléatoire, qui est le bruit. En utilisant le mode soustraction, la forme est creusée par le bruit paramétré auparavant.





Marble & Ball

Annexes p.79

Dans ces exemples, l'idée est d'utiliser des formes simples pour ainsi y creuser à l'aide du bruit. Le résultat donne alors quelque chose d'organique et aléatoire, avec des aspérités.

Pour donner l'illusion que le centre de la forme est différent de sa surface, il faut sélectionner tous les polygones qui sont « à l'intérieur » de cette dernière, et ensuite attribuer un nouveau matériau.

David

Annexes p.80

Dans cet exemple, il n'est plus question d'objets 3D simples, mais d'appliquer cette méthode sur des formes plus complexes. La démarche reste similaire, mais il s'agit ici d'une statue scannée en 3D (la tête de David par Michel-Ange).

Lorsque la tête est creusée, l'intérieur est visible et laisse alors apparaître un crâne en or.

ANIMATIONS ALÉATOIRES

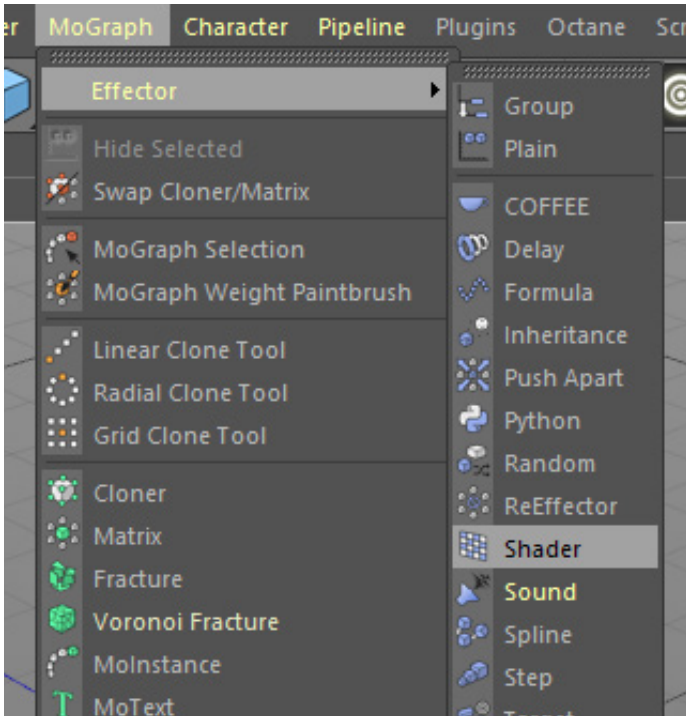
Pour ce dernier chapitre concernant la 3D, le principe est d'animer des objets de manière aléatoire, sur base d'un bruit. Ce bruit n'est donc pas visible concrètement comme les exemples précédents, mais agit plutôt comme un champ de valeurs quelconques qui affecte la position, la rotation, l'échelle, etc.

HUD

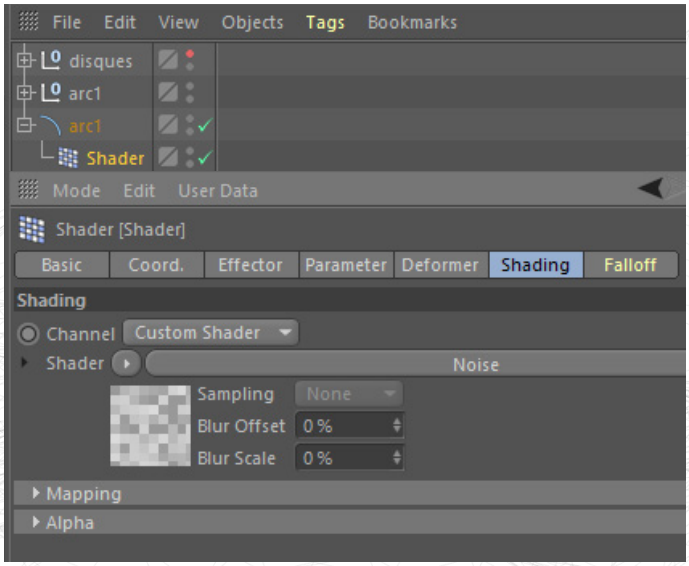
Annexes p.81

Pour cet exemple, l'animation est décomposée en plusieurs parties, qui seront unies en un projet final regroupant tous ces éléments.

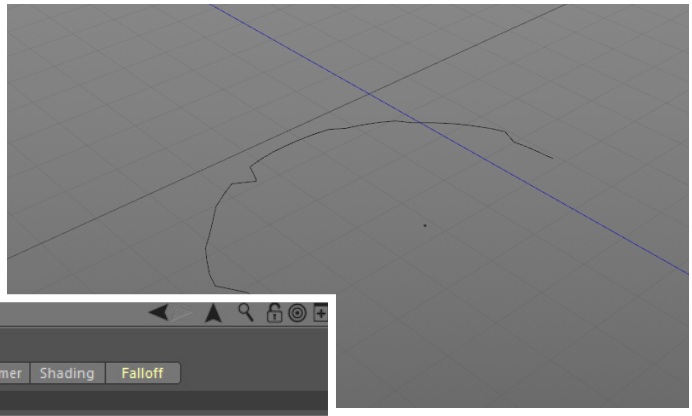
En premier lieu, il s'agit d'animer une courbe selon le bruit. Il n'est pas question ici de « displacer », mais de « shader », un autre outil du MoGraph de Cinema4D.

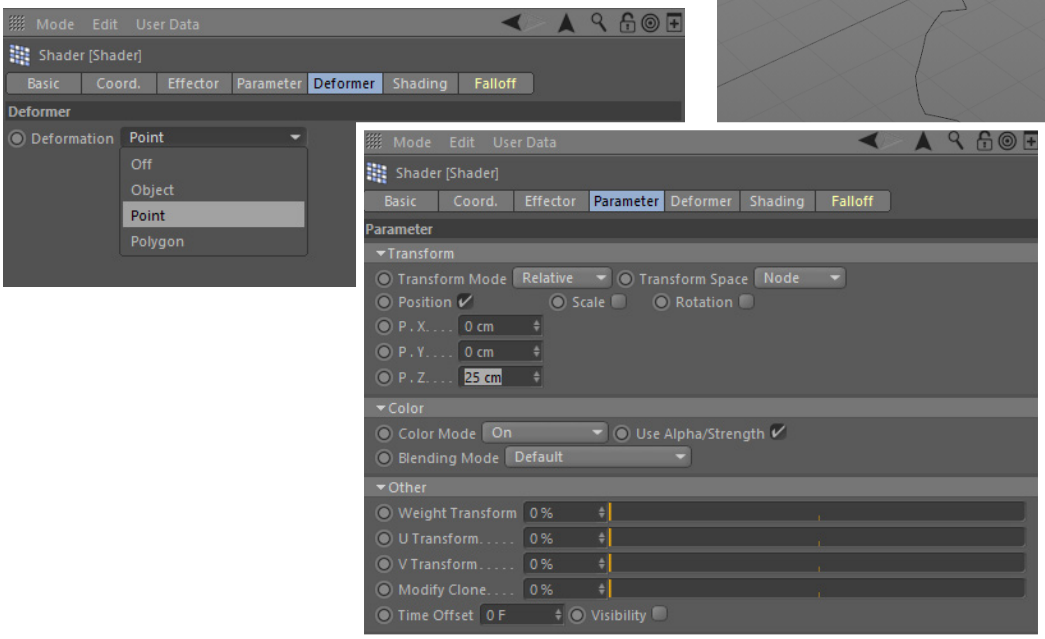


Dans les paramètres de ce « shader » est appliqué un bruit qui sera la source du mouvement. Pour ce projet, le « Cell Noise » sera approprié.



Reste à déterminer ce qu'affecte le bruit. Dans le cas de cette courbe, il s'agit de la position des points de cette courbe selon l'axe Z. Pour que la déformation prenne effet, il s'agit dans un second temps d'affecter le « shader » aux points de la courbe, et non pas à l'objet.





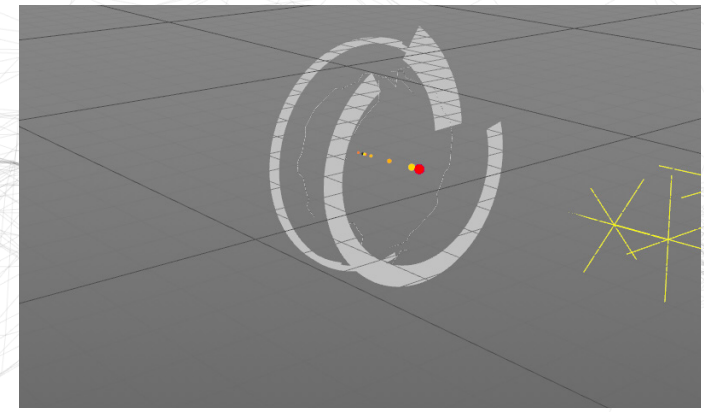
C'est en attribuant enfin une vitesse au bruit que l'animation se crée. Le « Cell Noise » a cette particularité d'avoir une animation très abrupte, ce qui est recherché dans ce projet.

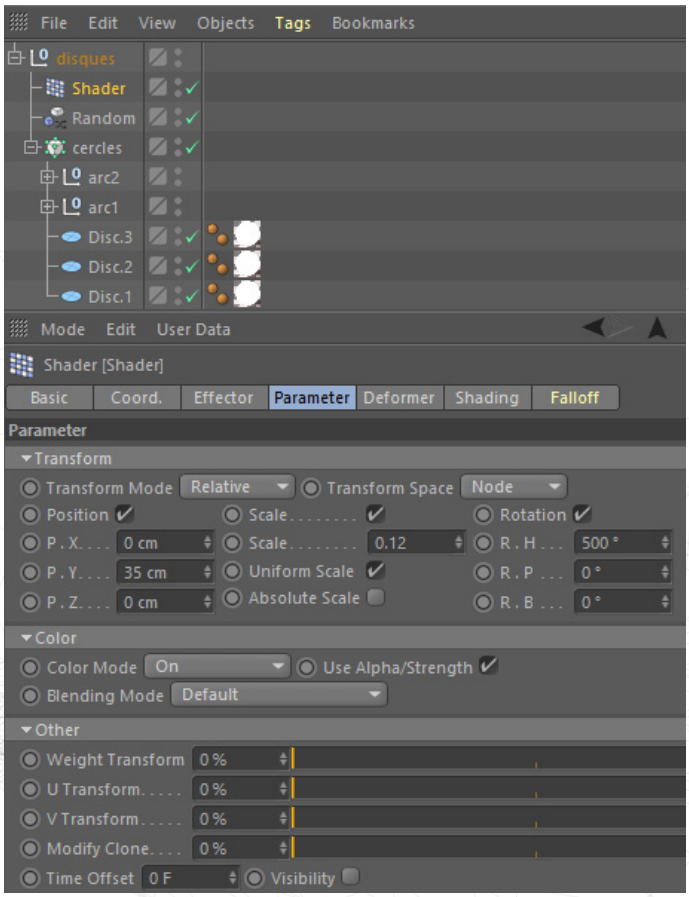
Animation courbe :
https://www.ludwigdejonckheere.com/noise/assets/animation/anim_courbe.mp4

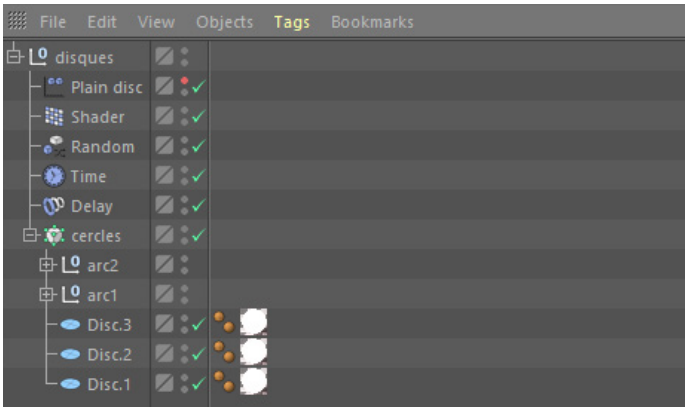
Vient ensuite l'animation d'arcs de cercle ainsi que de la courbe précédente. À cette fin, ces différents éléments sont entreposés dans un « cloner ». C'est sur ce dernier qu'un nouveau « shader » prend effet.

Il s'agit ici d'utiliser la même méthode que pour la courbe, à la différence que l'animation affecte à la fois la position sur un axe, l'échelle globale, ainsi que la rotation selon un axe.

Animation courbe :
https://www.ludwigdejonckheere.com/noise/assets/animation/anim_courbe.mp4

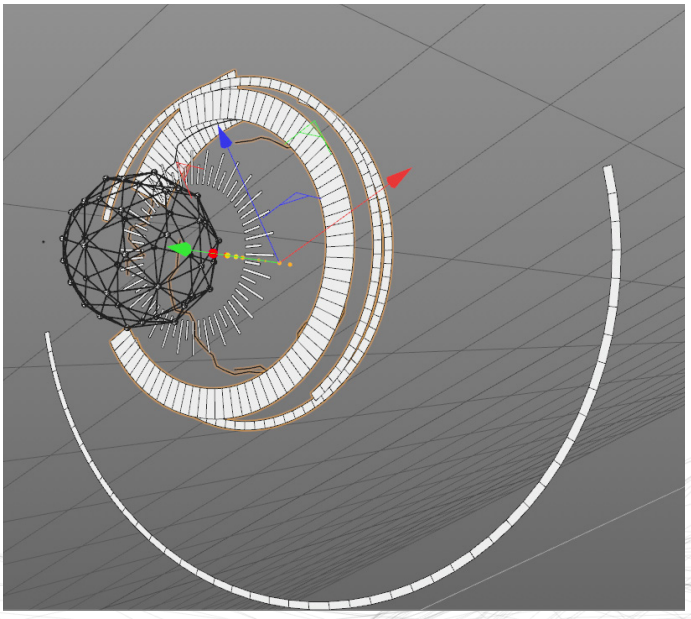


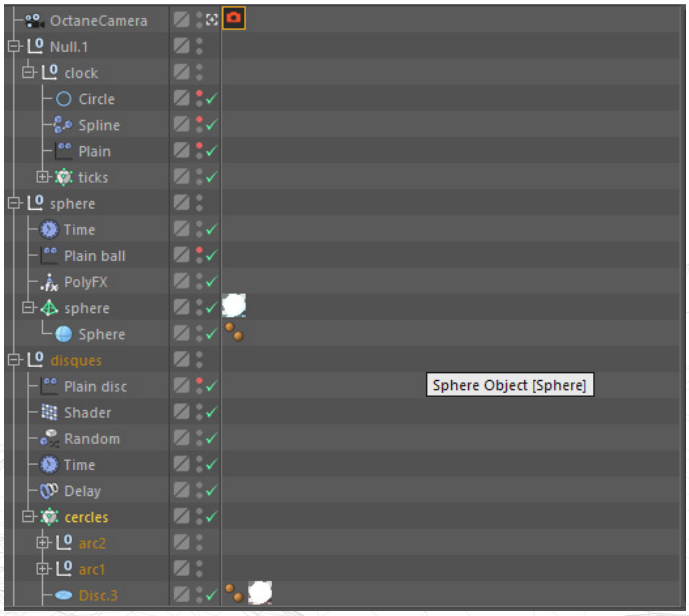




Pour enjoliver le tout, différents outils du « MoGraph » sont appliqués en aval, que ce soit pour rendre l'animation moins brutale lors mouvement, ou encore l'apparition des éléments de manière esthétique, ou enfin de permettre une rotation continue des éléments sans avoir recours aux clés d'animation.

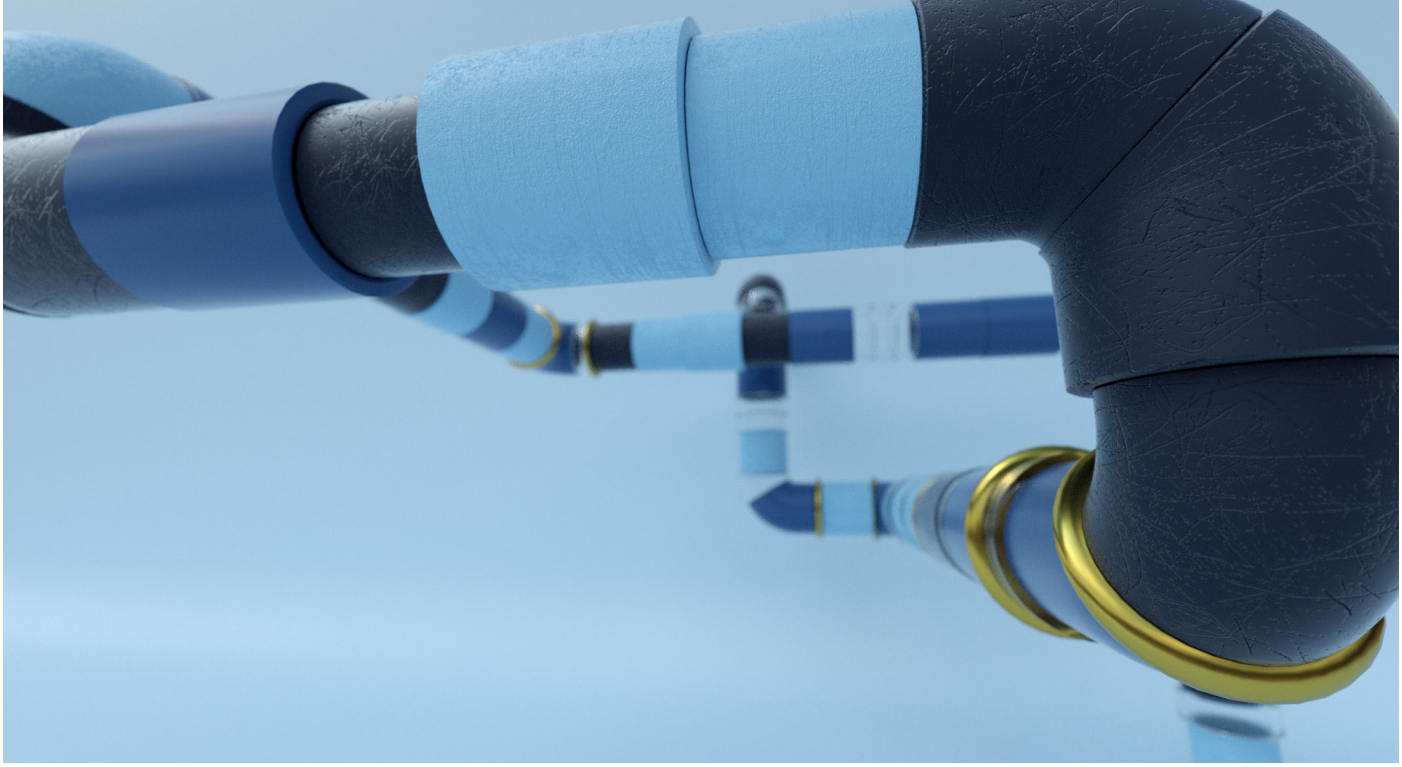
L'animation finale reprend alors les exemples précédents, ainsi que quelques objets 3D ajoutés pour donner du détail supplémentaire à l'animation.





Animation Élements :
https://www.ludwigdejonckheere.com/noise/assets/animation/anim_no_track.mp4

Animation HUD :
https://www.ludwigdejonckheere.com/noise/assets/animation/tracking_v2.mp4

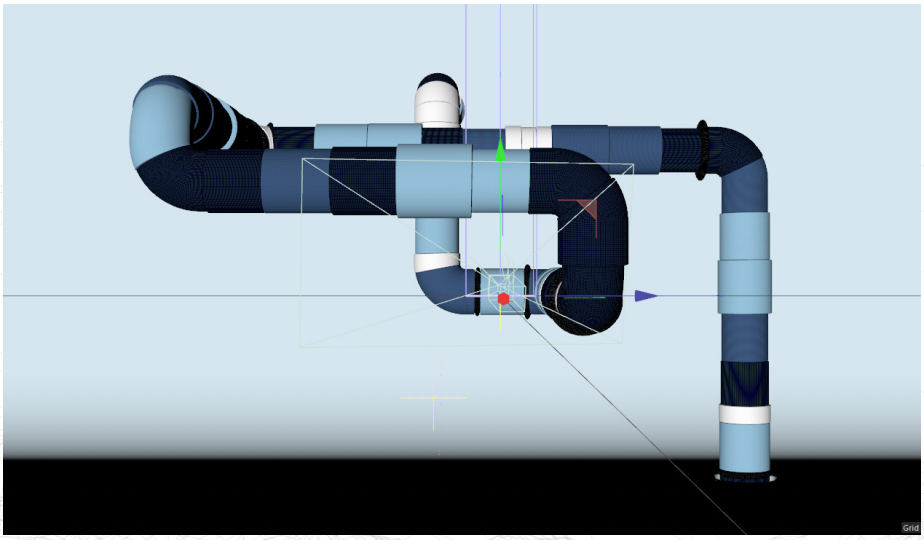


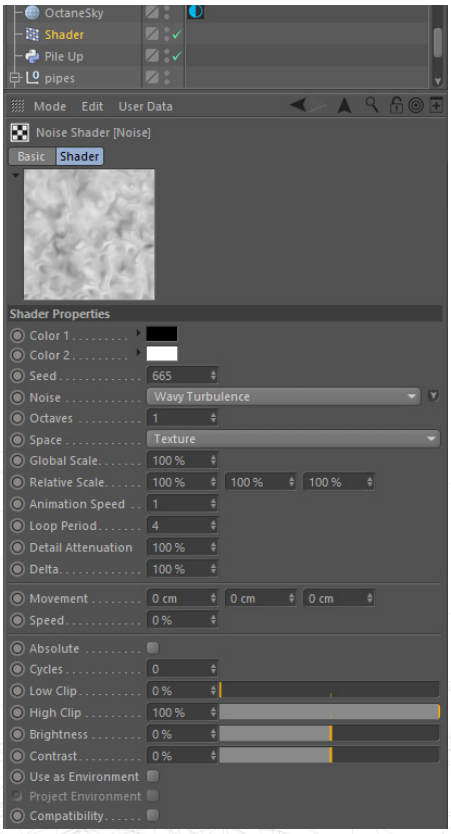
Tubes

Annexes p.81

Dans cet exemple, la méthode reste la même, à savoir l'utilisation du « shader » sur un « cloner ». À la différence que dans ce projet, il s'agit uniquement d'affecter l'échelle de chaque élément.

Ce projet consiste en un tube dupliqué via le « cloner », dont le « shader » affecte l'échelle, grâce à un bruit, ici le « Wavy Turbulence ». Chaque tube voit alors son échelle varier et donne alors naissance à une animation aléatoire.





Animation Tubes : https://www.ludwigdejonckheere.com/noise/assets/tubes_anim.mp4

SITE WEB

L'objectif final est de créer un site didactique reprenant les différents chapitres ci-dessus. Ce site est entièrement codé manuellement, en HTML, CSS et Javascript.

Ce site est néanmoins destiné à un public maitrisant un minimum Cinema4D et Octane, et qui désire en apprendre davantage sur le bruit et ses usages.

L'outil principal utilisé est Visual Studio Code, qui est complet en termes de code, et présente certaines extensions intéressantes. Parmi elles :

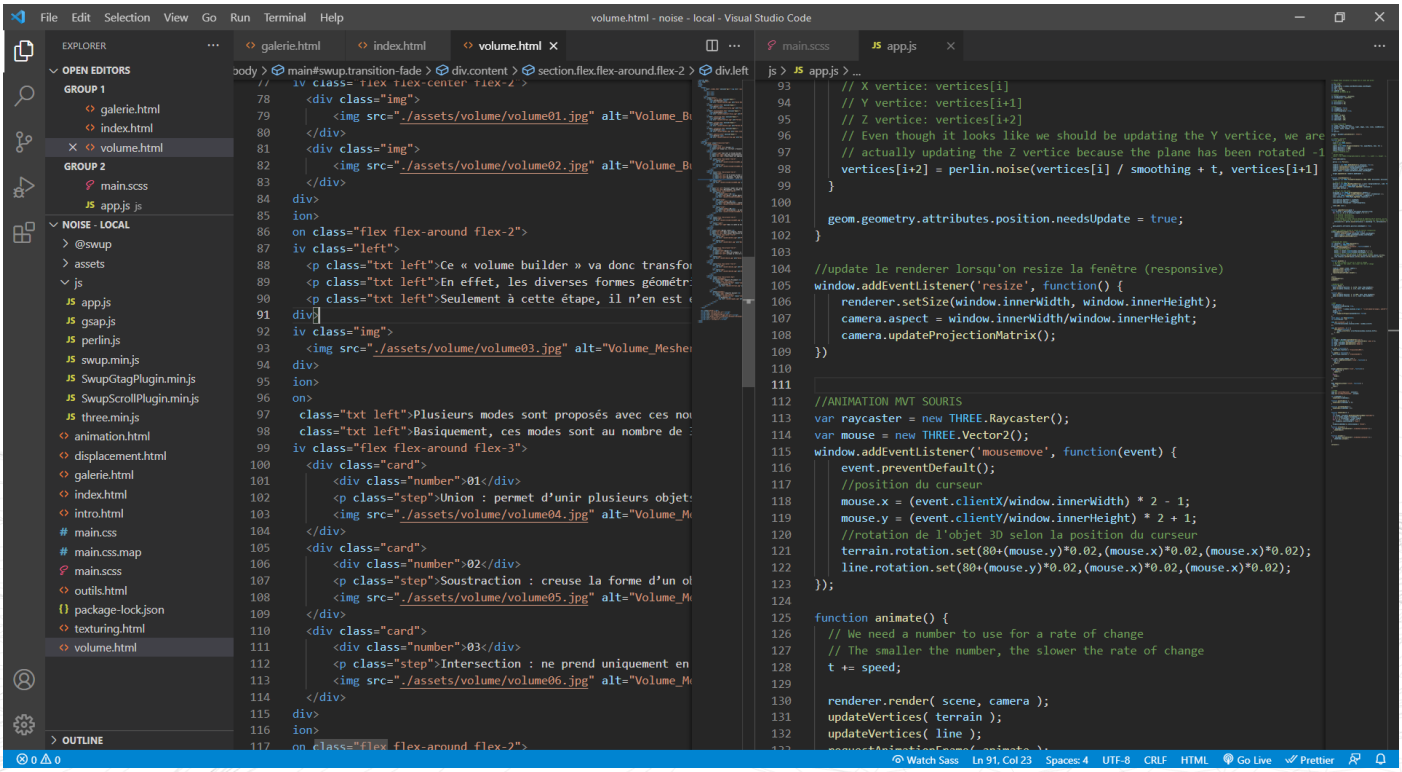
- « **Live Server** » : permet d'actualiser directement le site lors d'une sauvegarde et simule un serveur local.

- « **Live Sass Compiler** » : compile directement le fichier SCSS en CSS lors d'une sauvegarde.

- « **Auto Rename Tag** » : si une balise HTML est modifiée, sa balise fermante change aussi.

- « **Minify** » : lorsque les fichiers CSS et Javascript sont terminés, Minify peut les condenser pour un poids plus léger et une meilleure rapidité de lecture, en supprimant les espaces et les commentaires.

Visual Studio Code est aussi très personnalisable et adopte un code couleur de base qui rend le code très lisible et clair.

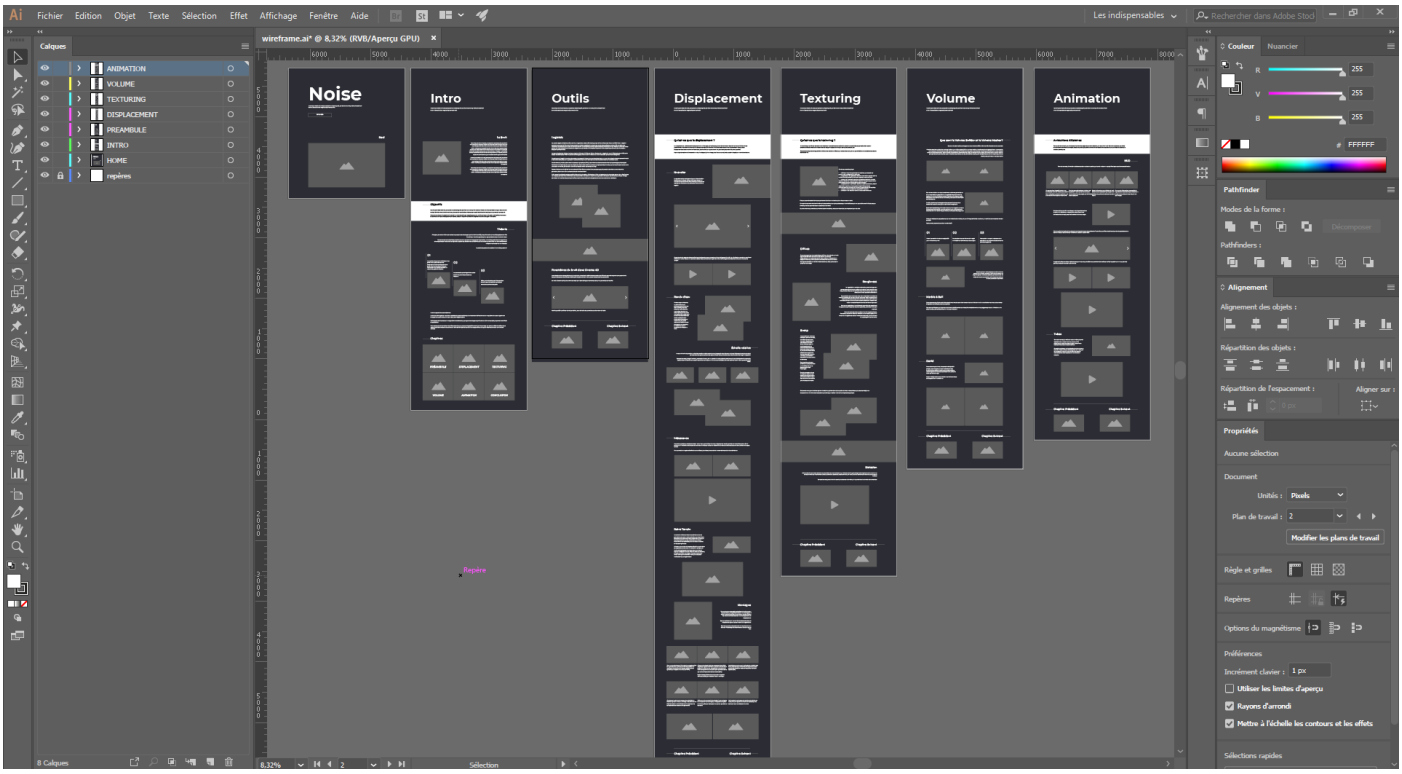



Noise
Et si le bruit jouait un rôle dans la 3D, qu'est-ce que cela donnerait ?
Plongez dans ce parcours démonstratif qui vous dévoile les facettes du bruit dans la création 3D. La créativité prend alors une tournure infinie.
EXPLORER
Reel
Noise
Et si le bruit jouait un rôle dans la 3D, qu'est-ce que cela donnerait ?
Plongez dans ce parcours démonstratif qui vous dévoile les facettes du bruit dans la création 3D. La créativité prend alors une tournure infinie.
EXPLORER
Reel

Wireframes

Annexes p.82

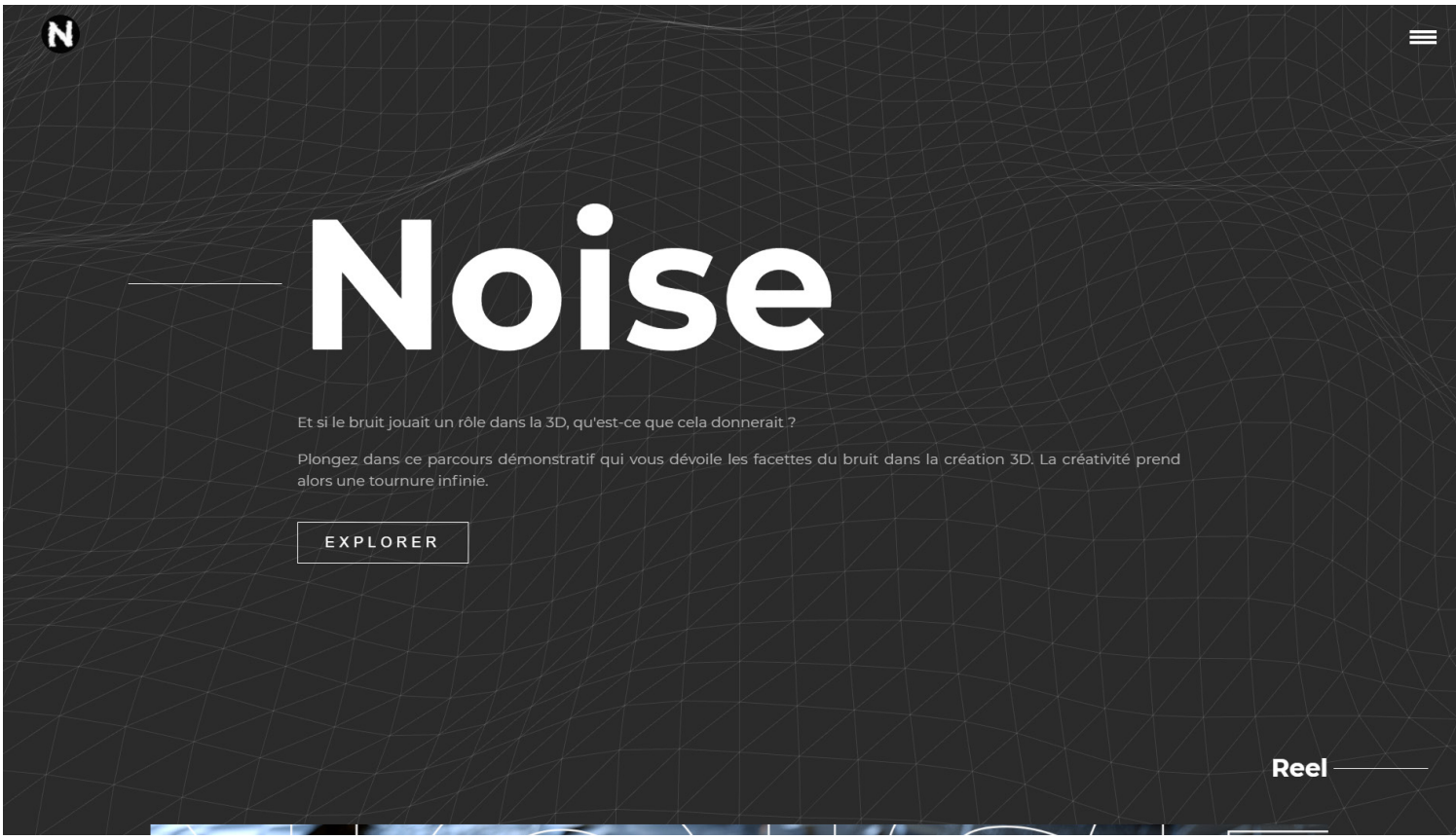
Avant de se lancer dans le code, il est important de créer un squelette du site. La création des différents wireframes a donc lieu sur Adobe Illustrator. Ces squelettes ne sont pas forcément respectés au pixel près, mais sont plutôt une idée globale de ce à quoi ressemblera le site par la suite.



Responsive

Il est de coutume de créer des wireframes pour les différentes largeurs d'écran. Cependant, dans le cas présent et étant donné la structure peu complexe du site, les wireframes pour tablettes, mobiles et petites largeurs n'ont pas lieu d'être.

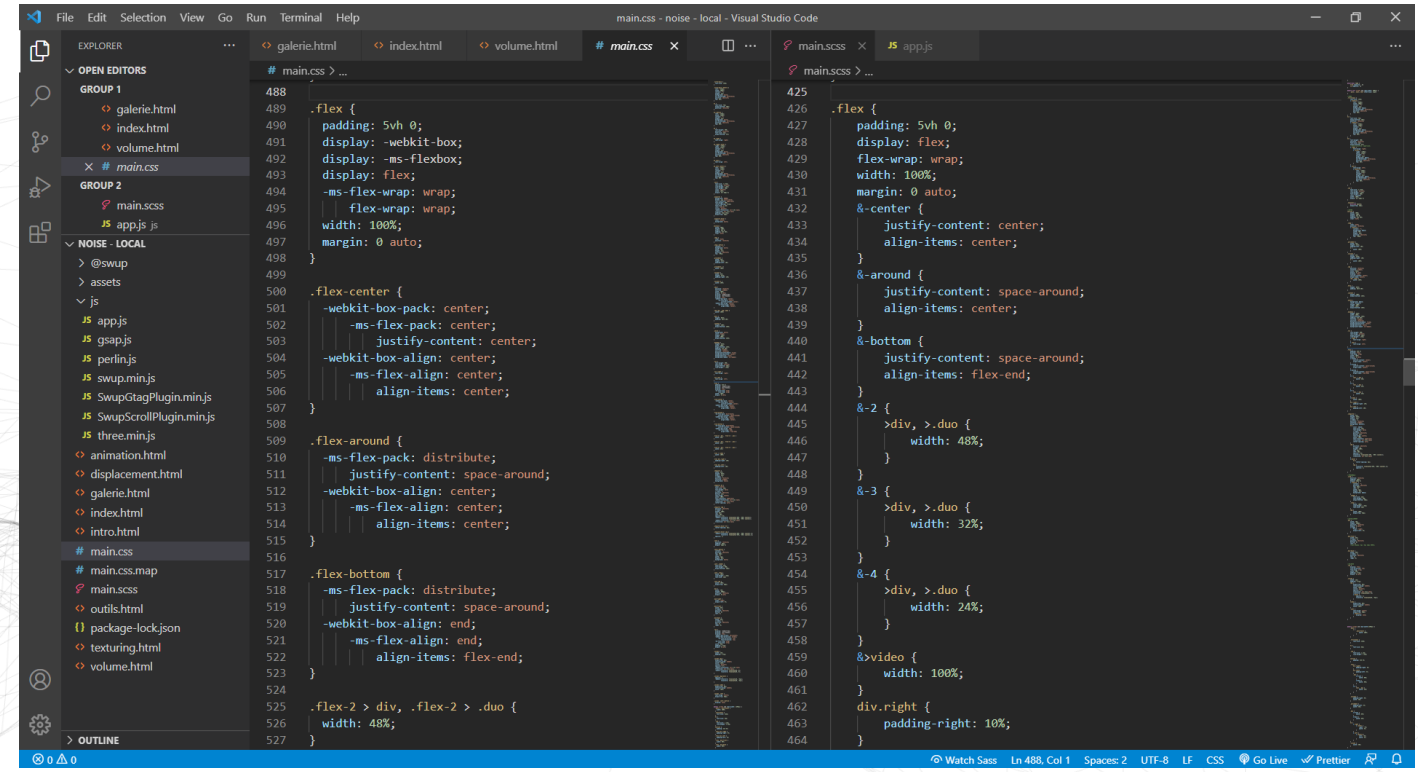
L'adaptation du site est directement réalisée via le code, où il ne s'agit que de changer quelques « flexbox » et largeurs d'éléments, sans complications.

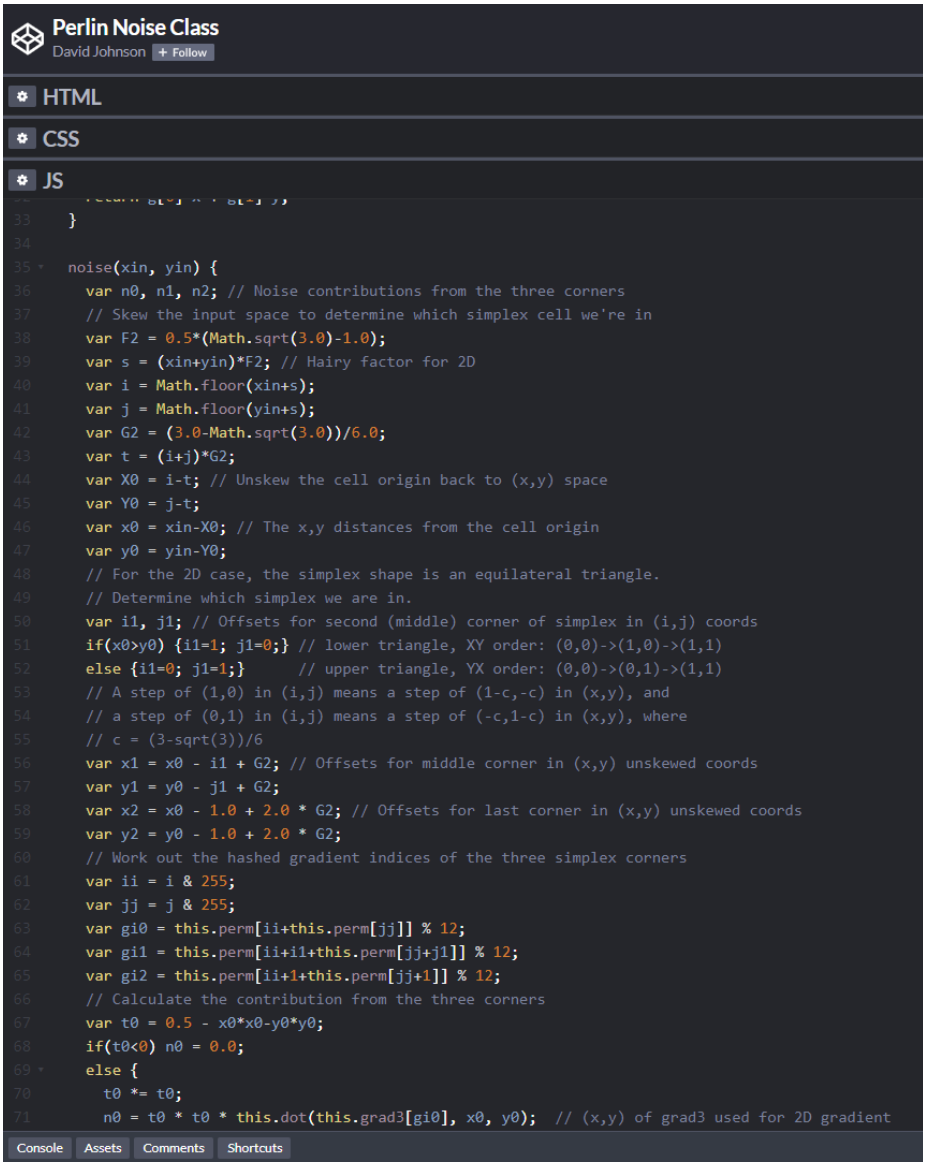


Code

La partie code du site n'a rien de spécial, hormis l'utilisation du SCSS. C'est une forme améliorée du CSS, où il est possible de créer des variables, des « mixins », ou encore de concaténer rapidement des classes ou balises html.

L'extension « Live SASS Compiler » compile alors le code SCSS dans un fichier CSS classique avec la bonne orthographe. Les variables sont transformées et les concaténations sont réalisées directement.





Three.Js

Afin d'adapter le site au thème concerné, à savoir le bruit dans la 3D, l'idée est d'avoir une 3D animée aléatoirement en fond, via un bruit. Le contenu du site est donc en avant-plan et n'est aucunement perturbé par l'animation qui a lieu en fond de site. Il faut aussi garder à l'esprit que cette animation ne doit pas être trop présente, au risque de trop capter l'attention de l'utilisateur.

Cette fonctionnalité est donc traitée grâce à Three.Js, une librairie Javascript qui permet d'intégrer de la 3D dans un navigateur, avec WebGL. L'animation consiste à animer une surface plane moyennement subdivisée, à laquelle un bruit agit sur les points de la géométrie, faisant monter ou descendre ces points.

Dans un premier temps, le bruit est généré depuis un algorithme. Ce dernier se résume en un fichier Javascript dans lequel se trouve une adaptation du code du bruit de Perlin.

L'algorithme du bruit ainsi que le code Javascript l'intégrant dans le navigateur avec Three.Js sont des codes repris de David Johnson, utilisateur du site CodePen, une plateforme permettant aux utilisateurs de créer et partager des bouts de codes.

Lorsque le script est intégré au code HTML, l'instance du bruit est ajoutée dans le fichier Javacript principal, et une fonction est appelée pour affecter ce bruit aux sommets de la géométrie.

```
var perlin = new Perlin();
```

```
function updateVertices(geom) {
  var vertices = geom.geometry.attributes.position.array;
  for ( var i = 0; i <= vertices.length; i += 3 ) {
    vertices[i+2] = perlin.noise(vertices[i] / smoothing + t, vertices[i+1] / smoothing + t) * amplitude;
  }

  geom.geometry.attributes.position.needsUpdate = true;
}
```

Après avoir intégré le script du bruit, la scène 3D peut être créée. Il s'agit d'une simple surface plane subdivisée. Pour le côté esthétique, il est question d'animer les arrêtes de la géométrie, et non pas les polygones. L'objet « PlaneGeometry » qu'offre la bibliothèque Three.Js n'est cependant pas adaptée puisqu'il faut ici exploiter les sommets de la surface, ce qui n'est pas possible avec cet objet. Il faut donc choisir l'option du « PlaneBufferGeometry », qui est une représentation du quadrillage, incluant polygones, sommets et arrêtes, et qui permet donc d'être exploité par le bruit de Perlin (il s'agit de déplacer les multiples sommets que composent la géométrie via le bruit).

Ses paramètres représentent respectivement la largeur, la longueur, le nombre de subdivisions en largeur et le nombre de subdivisions en longueur. Plus ces derniers nombres sont élevés, plus la géométrie est lisse.

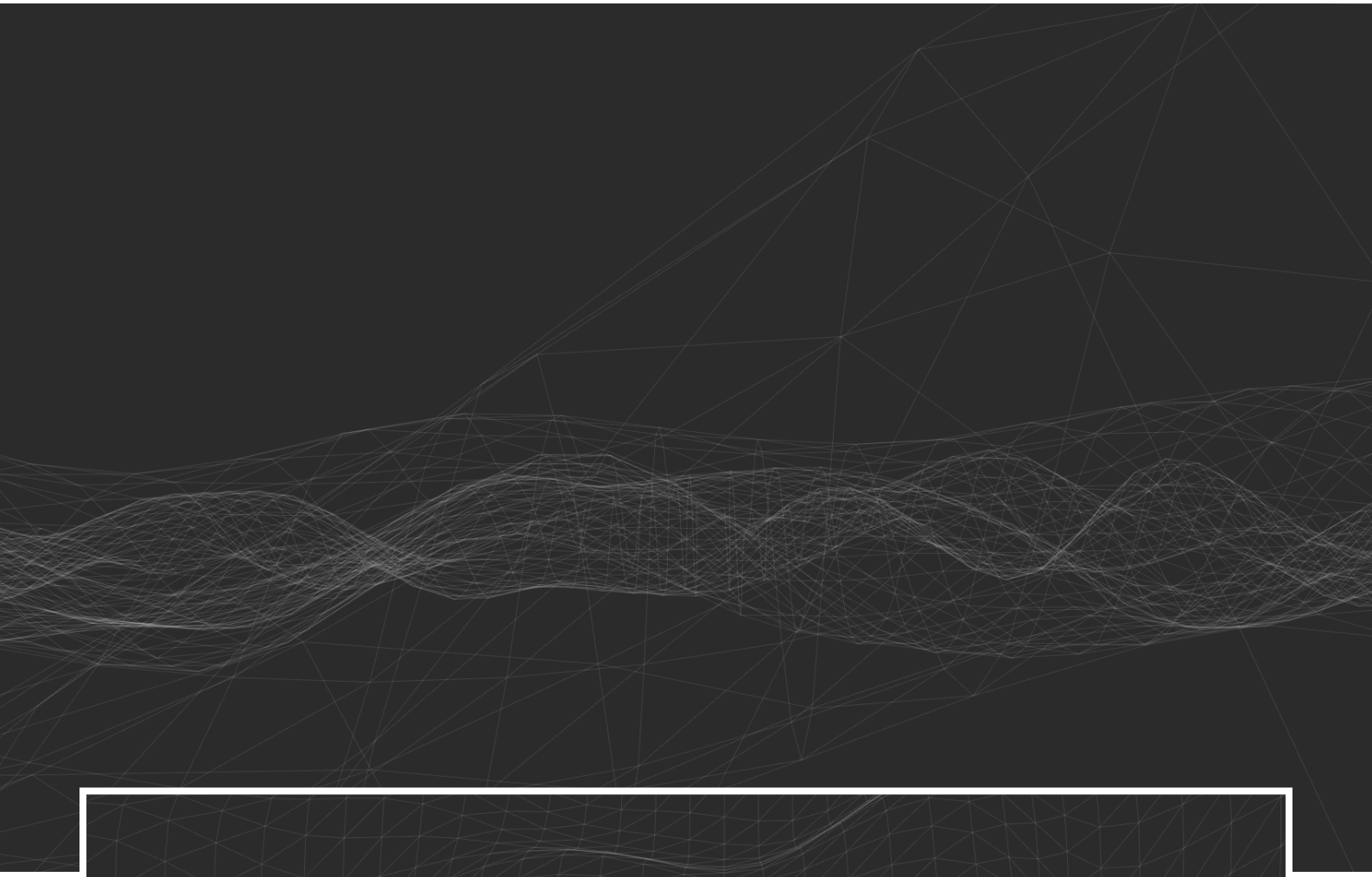
Le wireframe est quant à lui généré depuis la fonction « WireframeGeometry ». C'est une sorte de seconde géométrie créée sur base de la surface plane.

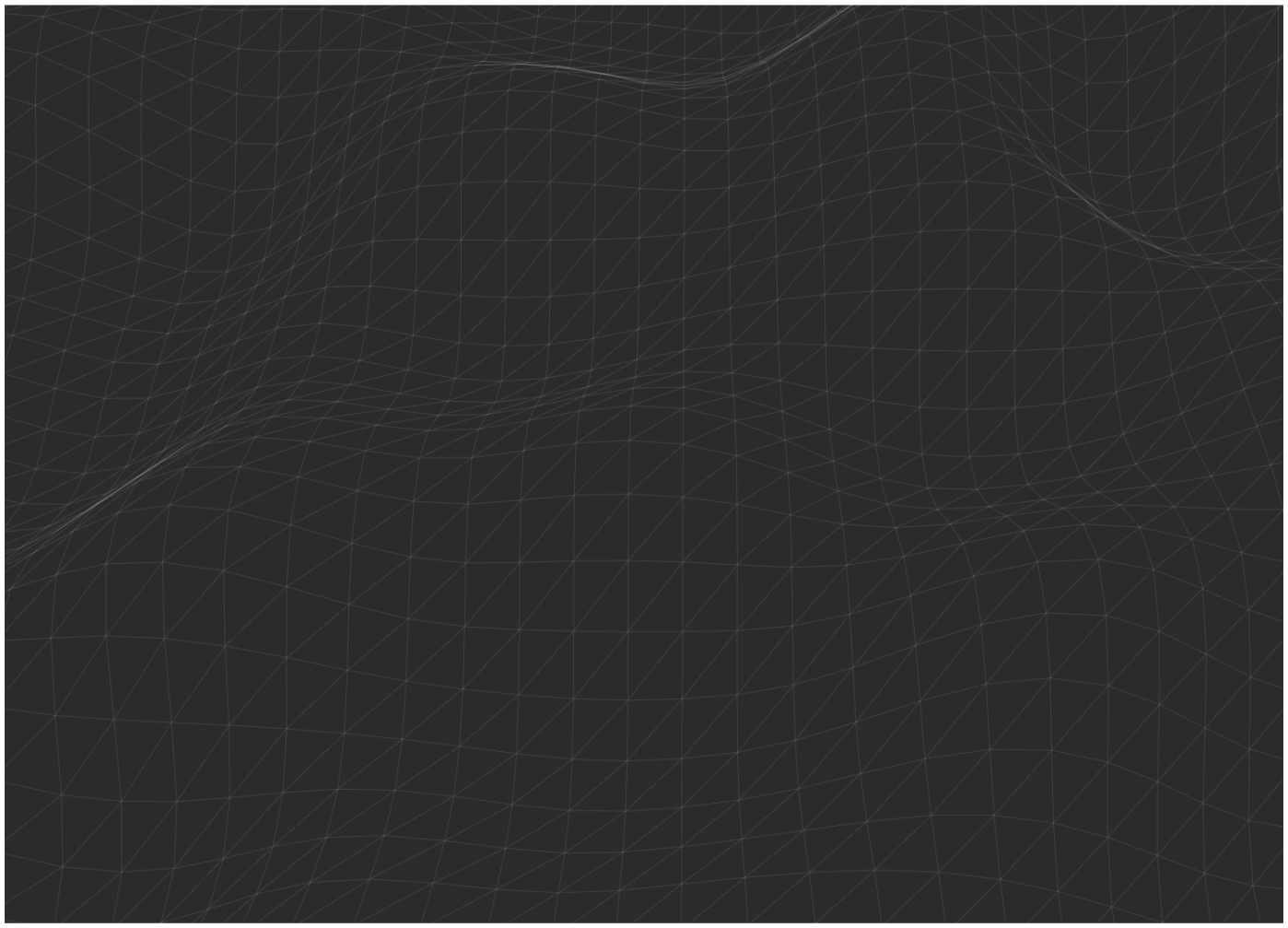
Afin de lui attribuer un matériau, il faut faire appel au « LineBasicMaterial », qui s'adapte uniquement aux wireframes.

Lorsque la géométrie et le matériau sont initialisés, il ne reste qu'à les unifier dans un « lineSegments ».

```
geometry = new THREE.PlaneBufferGeometry( 2000, 2000, divisionsX, divisionsY );
```

Le résultat rend alors une sorte de vague rendue en wireframe, directement dans le navigateur.





SWUP

La librairie Javascript SWUP permet des transitions de pages web sans rafraichissement, un peu à la manière de l'AJAX. Cette librairie prend en compte la navigation via l'historique (boutons « précédent » et « suivant » du navigateur). Elle gère aussi la mise en cache et les autres scripts Javascript présents sur le site.

Le fonctionnement est simple ; il suffit de donner un ID spécifique à une balise HTML pour qu'elle soit reconnue par le script SWUP (ici l'ID #swup). Cet ID doit être présent sur les différentes pages HTML dont le contenu doit être modifié.

Dans ce cas-ci il est question de changer le contenu de toute la page, hormis la navigation et le canvas qui contient la 3D en fond. Le contenu HTML de chaque page doit donc logiquement se trouver dans cette balise SWUP, et la navigation ainsi que le canvas ne possède pas cet ID, dans le but de rester à l'écran..

Différentes options sont aussi disponibles, comme le scroll en haut de chaque nouvelle page lors de la transition, la mise en cache ou non, l'animation des transitions lors du clic sur le bouton « précédent » ou « suivant », etc.

```
<!DOCTYPE html>
<html lang="fr" class="swup-enabled">
▶ <head>…</head>
▼ <body> == $0
  ▼ <header class="fade"> flex
    ▶ <a href="/noise/index.html" onclick="up()">…</a>
    ▶ <nav>…</nav>
    ▶ <div class="menu">…</div> flex
    </header>
  ▶ <main id="swup" class="transition-fade" data-swup="0">…</main>
  ▼ <div class="three">
      <canvas width="1254" height="980" style="display: block; width: 1254px; height: 980px;
      ">
    </div>
    <script src="./js/three.min.js"></script>
    <script src="./js/swup.min.js"></script>
    <script src="./js/SwupScrollPlugin.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.6.1/gsap.min.js"></script>
    <script src="./js/perlin.js"></script>
    <script src="./js/app.min.js"></script>
  </body>
</html>
```

```
//SLIDER
swup.on('contentReplaced', showSlides);
swup.on('willReplaceContent', unloadCarousel);

var slideIndex = 1;
showSlides(slideIndex);

function plusSlides(n) {
  showSlides(slideIndex += n);
}

function currentSlide(n) {
  showSlides(slideIndex = n);
}

function showSlides(n) {
  var i;
  var slides = document.getElementsByClassName("mySlides");
  if (n > slides.length) {slideIndex = 1}
  if (n < 1) {slideIndex = slides.length}
  for (i = 0; i < slides.length; i++) {
      slides[i].style.display = "none";
  }
  slides[slideIndex-1].style.display = "block";
}

function unloadCarousel() {
  if (document.querySelector('.slideshow-container')) {
    showSlides.unload();
  }
}

showSlides();
```

Certaines pages contiennent des carrousels en Javascript. De ce fait, ces fonctionnalités entrent en collision avec SWUP. En effet, le script d'un carrousel s'exécute lors de l'arrivée sur la page. Cependant, si l'utilisateur quitte cette dernière et y revient plus tard, le carrousel n'est plus fonctionnel.

La solution est de « décharger » le script du carrousel chaque fois que l'utilisateur quitte la page, et le charge à nouveau lors de son retour sur celle-ci.

Le carrousel est donc stocké dans une fonction propre, et une seconde fonction de déchargement du carrousel est initiée. La librairie SWUP prend en charge le reste grâce à des fonctions spécifiques.

GSAP

GSAP, ou GreenSock Animation Plugin, est une librairie Javascript facilitant des animations complexes d'un site, que ce soit au niveau des propriétés CSS ou, dans cet exemple, un mouvement de caméra de la 3D lors d'un clic sur un lien.

```
//ANIMATION GSAP
function down() {
  gsap.to(camera.position, 3, {y:80, ease: Expo.easeOut});
  gsap.to(camera.rotation, 3, {x:0, ease: Expo.easeOut});
};

function up() {
  gsap.to(camera.position, 3, {y:1000, ease: Expo.easeOut});
  gsap.to(camera.rotation, 3, {x:-1, ease: Expo.easeOut});
};
```

Javascript

L'intégration du bruit de Perlin ainsi que de la 3D animée sont des codes repris de David Johnson sur CodePen. De plus, les scripts de carrousels sont des adaptations de codes repris sur W3School. Certains éléments de ces carrousels ont été supprimés ou adaptés au site. Hormis ceci, les autres fonctionnalités sont codées à la main de manière personnelle.

Parmi ces fonctionnalités, il y a premièrement la fonction du menu qui apparait et disparait à droite de l'écran. Après avoir récupérer les éléments HTML nécessaires à la fonction du menu, il s'agit simplement de créer une fonction qui se base sur le modulo. En effet, un compteur est initialisé à 0. Si le modulo 2 de ce compteur vaut 0, le menu apparait. Si son modulo est différent de 0, le menu se rétracte.

```
//MENU
var burger = document.querySelector('nav');
var links = Array.from(document.querySelectorAll('.menu a'));
var menu = document.querySelector('.menu');
var home = document.querySelector('#home');
var nbr = 0;

var hide = function() {
  menu.style.transform = "translateX(100%)";
}
var appear = function() {
  menu.style.transform = "translateX(0%)";
}

//clic sur le hamburger
burger.addEventListener('click', function() {
  if (nbr%2==0) {
    appear();
  }
  else {
    hide();
  }
  nbr++;
})

//clic sur un lien de la nav
for (i=0; i<links.length; i++) {
  links[i].addEventListener('click', function() {
    nbr = 0;
    hide();
  })
}

//clic sur le logo Home
home.addEventListener('click', function() {
  nbr = 0;
  hide();
})
```

Lors du clic sur le hamburger, le menu apparait en premier lieu étant donné la valeur 0 du nombre, suivi de l'incrémentation de ce dernier, qui vaut alors 1. Cette fonction est utile si le bouton hamburger est cliqué plusieurs fois de suite. Le compteur augmente donc, avec à le menu qui apparait un nombre sur 2.

Si un lien est ensuite cliqué, le compteur est réinitialisé à 0, et le menu disparait. Ceci a pour effet d'avoir un compteur neuf lors de l'arrivée de l'utilisateur sur une nouvelle page. Cette fonction est aussi attribuée au logo d'accueil du site. En ce qui concerne le menu, le script comprend une boucle, pour affecter la fonction à chaque lien que composent le menu.

Une animation de la 3D est aussi créée lorsque le curseur bouge. La 3D en fond s'oriente alors en fonction de la position du curseur. Ceci est rendu possible grâce à la fonction de Raycaster de « Three.js », qui demande aussi la fonction « Vecter2 ». Cependant, ceci rend impossible la sélection du texte sur le site.

```
//ANIMATION MVT SOURIS
var raycaster = new THREE.Raycaster();
var mouse = new THREE.Vector2();
window.addEventListener('mousemove', function(event) {
    event.preventDefault();
    //position du curseur
    mouse.x = (event.clientX/window.innerWidth) * 2 - 1;
    mouse.y = (event.clientY/window.innerHeight) * 2 + 1;
    //rotation de l'objet 3D selon la position du curseur
    terrain.rotation.set(80+(mouse.y)*0.02,(mouse.x)*0.02,(mouse.x)*0.02);
    line.rotation.set(80+(mouse.y)*0.02,(mouse.x)*0.02,(mouse.x)*0.02);
});
```

Enfin, le dernier script correspond à la « lightbox » présente sur la galerie. Cette fonction permet à l'utilisateur d'agrandir l'image cliquée, et ensuite naviguer sur cette « lightbox » via des boutons. Cette fonctionnalité est en partie reprise du carrousel de base. Il s'agit donc d'ouvrir un carrousel dans une « lightbox ».

En premier lieu, il faut donc créer un conteneur en fin de DOM qui sera la visionneuse. Ce conteneur contient une DIV qui réceptionnera par la suite les images, ainsi que les 2 boutons de navigation, et un bouton de fermeture permettant de quitter la « lightbox ».

```
<section class="lightbox fade">
    <div class="close fade">&times;</div>
    <div class="lightbox-slideshow">
        <div class="lightbox-mySlides fade"></div>
        <a class="prev">&#10094;</a>
        <a class="next">&#10095;</a>
    </div>
</section>
```

L'ensemble de la section est désactivée grâce au « display : none ; », et sera changée en « display : flex ; » lors du clic sur une image de la galerie (« flex » et non pas « block » pour pouvoir center le conteneur de l'image directement dans la visionneuse). Après avoir établi ceci et créer le CSS qui est en relation vient le script.

L'idée est que, lors du clic sur une image de la galerie :

1) Un clone de cette image est créé

2) Ce clone est ajouté dans le conteneur de la visionneuse

3) La navigation en haut de l'écran disparait

4) La visionneuse apparait

Pour que la fonction prenne cours sur chaque image et vidéo de la galerie, cette fonction est initialisée dans une boucle dont le compteur maximum est le nombre d'éléments que compte la galerie.

```
function lightbox() {
  //tableau contenant les images et vidéos de la galerie
  var galerieImg = Array.from(document.querySelectorAll('.column img, .column video'));
  //lightbox
  var lightbox = document.querySelector('.lightbox');
  //conteneur réceptionnant les images dans la lightbox
  var lightboxInner = document.querySelector('.lightbox .lightbox-mySlides');
  //boutons
  var before = document.querySelector('.lightbox .prev');
  var after = document.querySelector('.lightbox .next');
  var close = document.querySelector('.close');

  for (var k=0; k<galerieImg.length; k++) {
    //clic sur une image
    galerieImg[k].addEventListener('click', function() {
      //désactivation de la navigation en haut de la page
      document.querySelector('header').style.display = "none";
      //récupération du l'index de l'image cliquée dans le tableau
      var galerieIndex = galerieImg.indexOf(this);
      //clonage de l'image
      var clone = galerieImg[galerieIndex].cloneNode();
      //ajout du clone dans le conteneur de la lightbox
      lightboxInner.appendChild(clone);

      //attribution de la largeur de l'image dans le conteneur
      clone.style.width = "100%";
      //apparition de la lightbox qui n'est plus en display:none;
      lightbox.style.display = "flex";
```

Vient ensuite les fonctions des boutons de navigation. Lorsque ceux-ci sont cliqué :

1) Le clone actuel est supprimé

2) L'index du tableau contenant les images et vidéos est changé (-1 pour le bouton précédent, +1 pour le bouton suivant)

3) Un nouveau clone est créé sur base du nouvel index

4) Ce clone est injecté dans le conteneur de la lightbox

À noter qu'une fonction intermédiaire est initialisée, qui permet de boucler si l'utilisateur arrive en fin de galerie et clique sur le bouton suivant, ou qu'il arrive en début de galerie et qu'il clique sur le bouton précédent.

```
//clic sur le bouton suivant
after.addEventListener('click', function() {
  //suppression du clone actuel
  lightboxInner.removeChild(clone);
  //fonction de boucle du carrousel
  if (galerieIndex == (galerieImg.length)-1) {
    galerieIndex = 0;
  }
  else {
    //incrémentation de l'index du tableau contenant les images et vidéos
    galerieIndex++;
  }
  //nouveau clone créé sur base du nouvel index
  clone = galerieImg[galerieIndex].cloneNode();
  //ajout de ce nouveau clone dans le conteneur de la lightbox
  lightboxInner.appendChild(clone);

  //nouvel attribut style pour le nouveau clone
  clone.style.width = "100%";
})
//clic sur le bouton précédent
before.addEventListener('click', function() {
  //suppression du clone actuel
  lightboxInner.removeChild(clone);
  //fonction de boucle du carrousel
  if (galerieIndex == 0) {
    galerieIndex = (galerieImg.length)-1;
  }
  else {
    //décrémentation de l'index du tableau contenant les images et vidéos
    galerieIndex--;
  }
  //nouveua clone créé sur base du nouvel index
  clone = galerieImg[galerieIndex].cloneNode();
  //ajout de ce nouveau clone dans le conteneur de la lightbox
  lightboxInner.appendChild(clone);

  //nouvel attribut style pour le nouveau clone
  clone.style.width = "100%";
})
```

Enfin, une dernière fonction prend lieu permettant de fermer la visionneuse lors du clic sur le bouton de fermeture. Lorsqu'elle est exécutée :

1) La navigation en haut de l'écran réapparait

2) La visionneuse disparait

3) Le clone actuel est supprimé

L'entièreté de cette fonction présente aussi sa fonction de « déchargement », pour pouvoir être compatible avec SWUP.

```
//clic sur le bouton de fermeture
close.addEventListener('click', function() {
  //réapparition de la navigation en haut de l'écran
  document.querySelector('header').style.display = "block";
  //disparition de la lightbox
  lightbox.style.display = "none";
  //suppression du clone dans le conteneur de la lightbox
  lightboxInner.removeChild(clone);
})
```

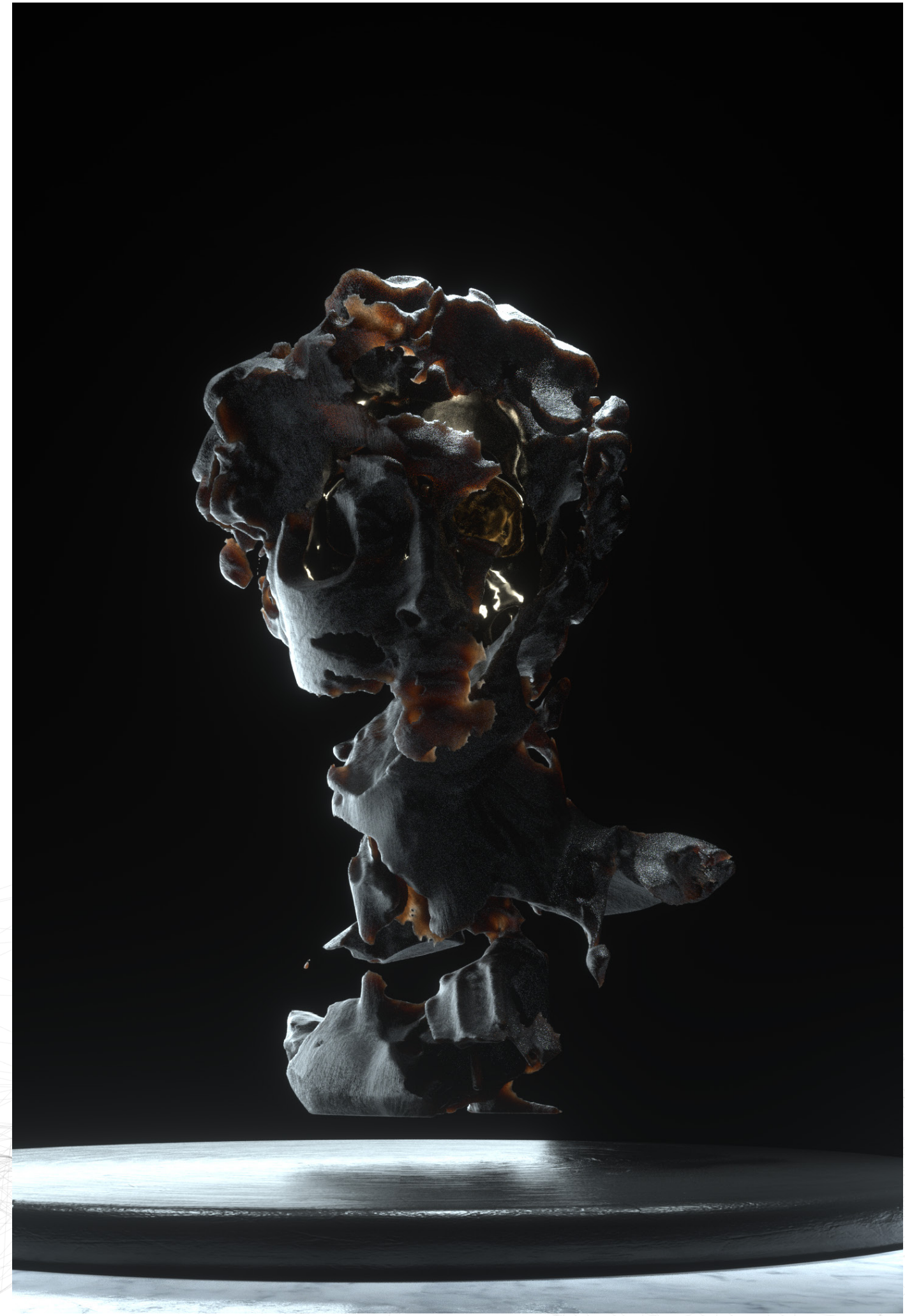
RÉSULTATS PRATIQUES

Le produit fini consiste donc en un site web didactique répertoriant ce rapport ainsi que ses différentes étapes. Il est disponible via l'adresse suivante :

https://ludwigdejonckheere.com/noise/index.html

Une vidéo de type «Showreel» répertoriant tous les projets vus précédemment a été conçue. Cette vidéo est disponible à l'adresse suivante :

https://www.ludwigdejonckheere.com/noise/assets/noise-reel.mp4



CONCLUS

ION

Le bruit est désormais un outil couramment utilisé et presque indispensable dans le monde de la création 3D, que ce soit dans un logiciel de 3D comme Cinema4D ou dans le web avec Three.Js par exemple.

Les possibilités qu'offre ce bruit sont infinies, seules les idées en sont sa limite. Ce travail met brièvement en lumière ses diverses utilisations, parce qu'il est impossible de couvrir complètement le sujet.

Cette innovation de Ken Perlin est révolutionnaire dans le domaine digital, mais aussi dans la création vidéoludique. En effet, nombre de jeux vidéo sont désormais générés de manière procédurale, grâce à des algorithmes se basant sur celui du bruit de Perlin.

Pour conclure, le bruit se présente comme un terrain de jeu, un bac à sable parfait pour toutes les expérimentations possibles. C'est un acteur important dans le monde du motion design, et un nouveau type de jeu qui se présente aux créateurs et motion designers. La recherche de formes ou d'animations générées aléatoirement n'a rien de plus amusante et satisfaisante.

CONTRAINTES ET DIFFICULTÉS RENCONTRÉES

Les contraintes rencontrées ont surtout été d'ordre techniques. En effet, certaines scènes 3D sont parfois un peu surchargée, et donne lieu à des ralentissements du logiciel. De plus, certains rendus sur Octane ont pris énormément de temps pour une animation de 4 secondes, malgré des paramètres parfois minimums.

Une machine datant de 5 ans fait le travail, mais il est certain qu'un ordinateur plus récent, notamment au niveau du processeur ou de la carte graphique, aurait facilité la tâche.

Une autre contrainte concerne les logiciels utilisés. En effet, certains plugins de Cinema4D auraient été les bienvenus : la dernière version d'Octane, ou encore d'autres plugins tiers comme Forester ou XParticles. Cependant, ces plugins ont un coût un peu onéreux en tant qu'étudiant.

Enfin, certaines difficultés ont été rencontrées, mais plutôt au niveau des connaissances. Certains fonctionnements sont parfois difficiles à comprendre sur Cinema4D, au niveau des éléments « MoGraph », et une maitrise moyenne du logiciel freine la créativité.

Il en va de même pour le code du site, où la création et la compréhension du code Javascript n'est pas chose aisée, ainsi que la maitrise de la librairie Three.Js.

Ce travail a permis d'approfondir certaines connaissances et de découvrir énormément de nouvelles choses.

BIBLIOGRAPHIE

THÉORIE

Bruit de Perlin In WIKIPEDIA. *L'encyclopédie libre* [en ligne]. 2021
Disponible sur : <https://fr.wikipedia.org/wiki/Bruit_de_Perlin>
[Page consultée le 12 mai 2021]

Perlin Noise In WIKIPEDIA. *L'encyclopédie libre* [en ligne]. 2021
Disponible sur : <https://en.wikipedia.org/wiki/Perlin_noise>
[Page consultée le 12 mai 2021]

Pseudo-aléatoire In WIKIPEDIA. *L'encyclopédie libre* [en ligne]. 2019
Disponible sur : <https://fr.wikipedia.org/wiki/Pseudo-aléatoire>
[Page consultée le 12 mai 2021]

Graine aléatoire In WIKIPEDIA. *L'encyclopédie libre* [en ligne]. 2016
Disponible sur : <https://fr.wikipedia.org/wiki/Graine_aléatoire>
[Page consultée le 12 mai 2021]

Bruit de gradient In WIKIPEDIA. *L'encyclopédie libre* [en ligne]. 2019
Disponible sur : <https://fr.wikipedia.org/wiki/Bruit_de_gradient>
[Page consultée le 12 mai 2021]

Gradient In WIKIPEDIA. *L'encyclopédie libre* [en ligne]. 2021
Disponible sur : <https://fr.wikipedia.org/wiki/Gradient>
[Page consultée le 12 mai 2021]

Path tracing In WIKIPEDIA. *L'encyclopédie libre* [en ligne]. 2020
Disponible sur : <https://fr.wikipedia.org/wiki/Path_tracing>
[Page consultée le 12 mai 2021]

Ray tracing In WIKIPEDIA. *L'encyclopédie libre* [en ligne]. 2021
Disponible sur : <https://fr.wikipedia.org/wiki/Ray_tracing>
[Page consultée le 12 mai 2021]

Cinema 4D In WIKIPEDIA. *L'encyclopédie libre* [en ligne]. 2020
Disponible sur : <https://fr.wikipedia.org/wiki/Cinema_4D>
[Page consultée le 12 mai 2021]

Octane (moteur de rendu) In WIKIPEDIA. *L'encyclopédie libre* [en ligne]. 2019
Disponible sur : <https://fr.wikipedia.org/wiki/Octane_(moteur_de_rendu)>
[Page consultée le 12 mai 2021]

Displacement mapping In WIKIPEDIA. *L'encyclopédie libre* [en ligne]. 2019
Disponible sur : <https://fr.wikipedia.org/wiki/Displacement_mapping>
[Page consultée le 12 mai 2021]

Khan Academy. *Le bruit de Perlin* [en ligne]. 2021
Disponible sur :<https://fr.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-noise/a/perlin-noise>
[Page consultée le 12 mai 2021]

MANULE UTILISATEUR D'OCTANE VERSION2019.1. *C4D Noise* [en ligne]. 2019
Disponible sur : <http://www.aoktar.com/octane/OCTANE%20HELP%20MANUAL.html?Noise1.html>
[Page consultée le 12 mai 2021]

LIBRAIRIES JAVASCRIPT

GreenSock. *Getting Started with GSAP* [en ligne]. 2019
Disponible sur : <https://greensock.com/get-started>
[Page consultée le 27 mai 2021]

SWUP. *Getting Started* [en ligne]. 2017
Disponible sur : <https://swup.js.org/getting-started>
[Page consultée le 27 mai 2021]

Github. *Jquery PJAX* [en ligne]. 2017
Disponible sur : <https://github.com/defunkt/jquery-pjax>
[Page consultée le 27 mai 2021]

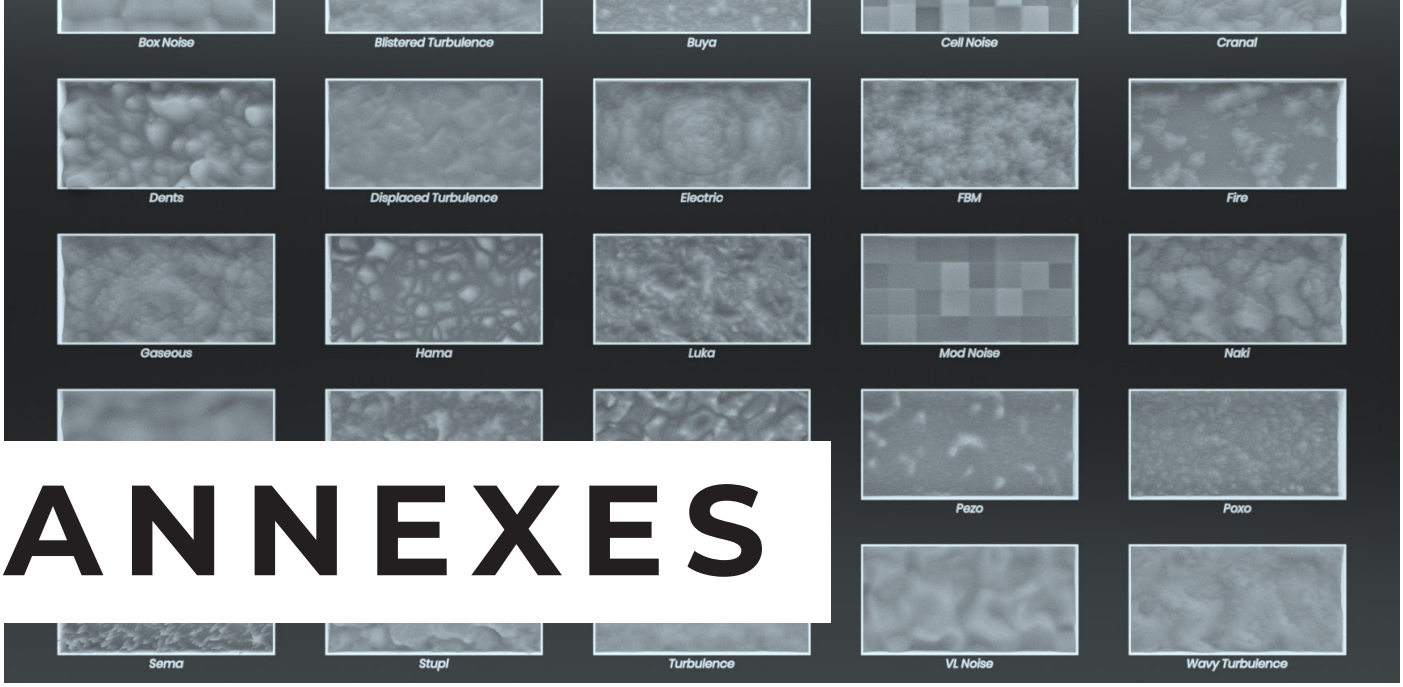
DOCUMENTATION DE THREE.JS. *Getting Started* [en ligne]. 2021
Disponible sur : <https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene>
[Page consultée le 27 mai 2021]

GÉNÉRATION DU BRUIT AVEC THREE.JS

David Johnson In CODEPEN. *Lunch and Learn : Wave* [en ligne]. 2021
Disponible sur : <https://codepen.io/struct/pen/pQRBjO>
[Page consultée le 27 mai 2021]

David Johnson In CODEPEN. *Perlin Noise Class* [en ligne]. 2018
Disponible sur : <https://codepen.io/struct/pen/bQgJmx?editors=0010>
[Page consultée le 27 mai 2021]

Victor Vergara In CODEPEN. *Perlin Noise* [en ligne]. 2018
Disponible sur : <https://codepen.io/vcomics/pen/djqNrm>
[Page consultée le 27 mai 2021]



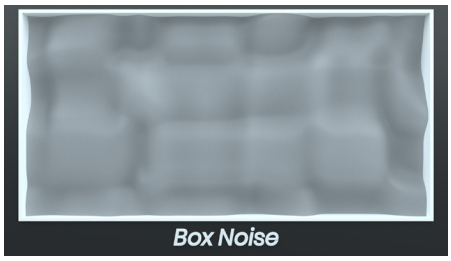
ANNEXES

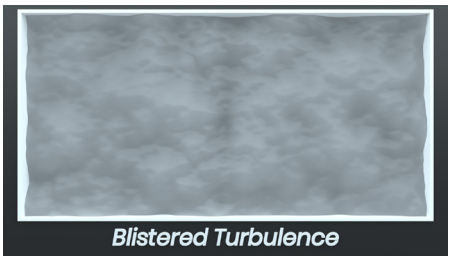
DISPLACEMENT

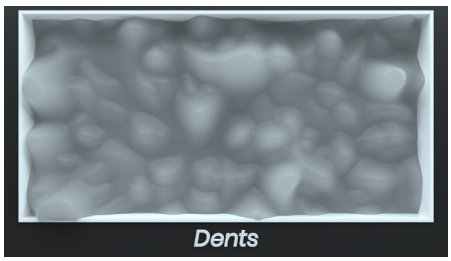
Nuancier

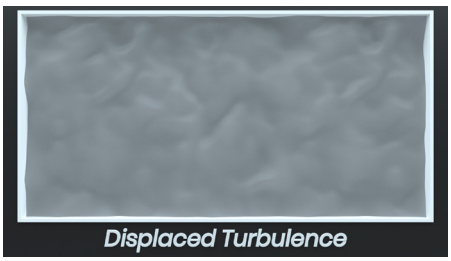
Mouvement Cell Noise :
https://www.ludwigdejonckheere.com/noise/assets/nuancier/cell_noise.mp4

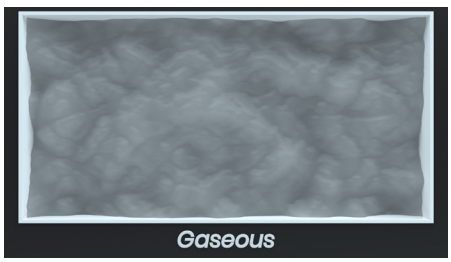
Mouvement Mod Noise :
https://www.ludwigdejonckheere.com/noise/assets/nuancier/mod_noise.mp4

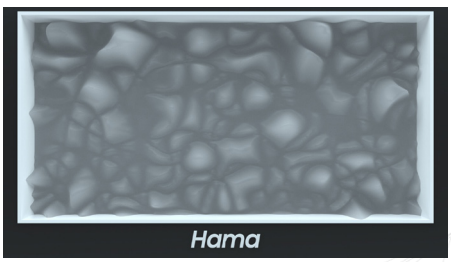


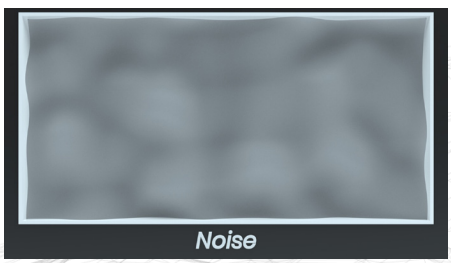


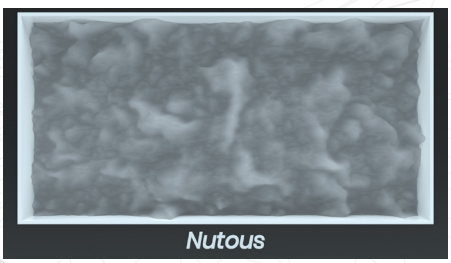


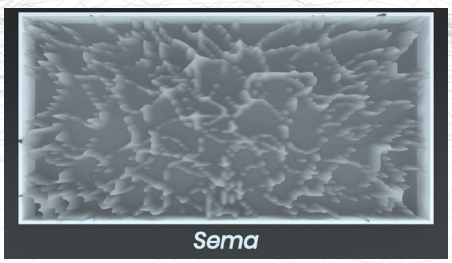


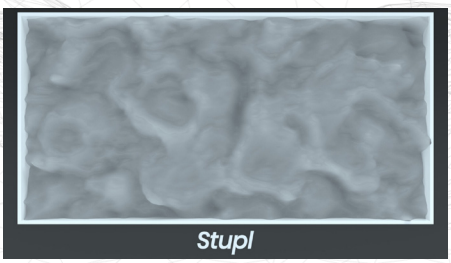


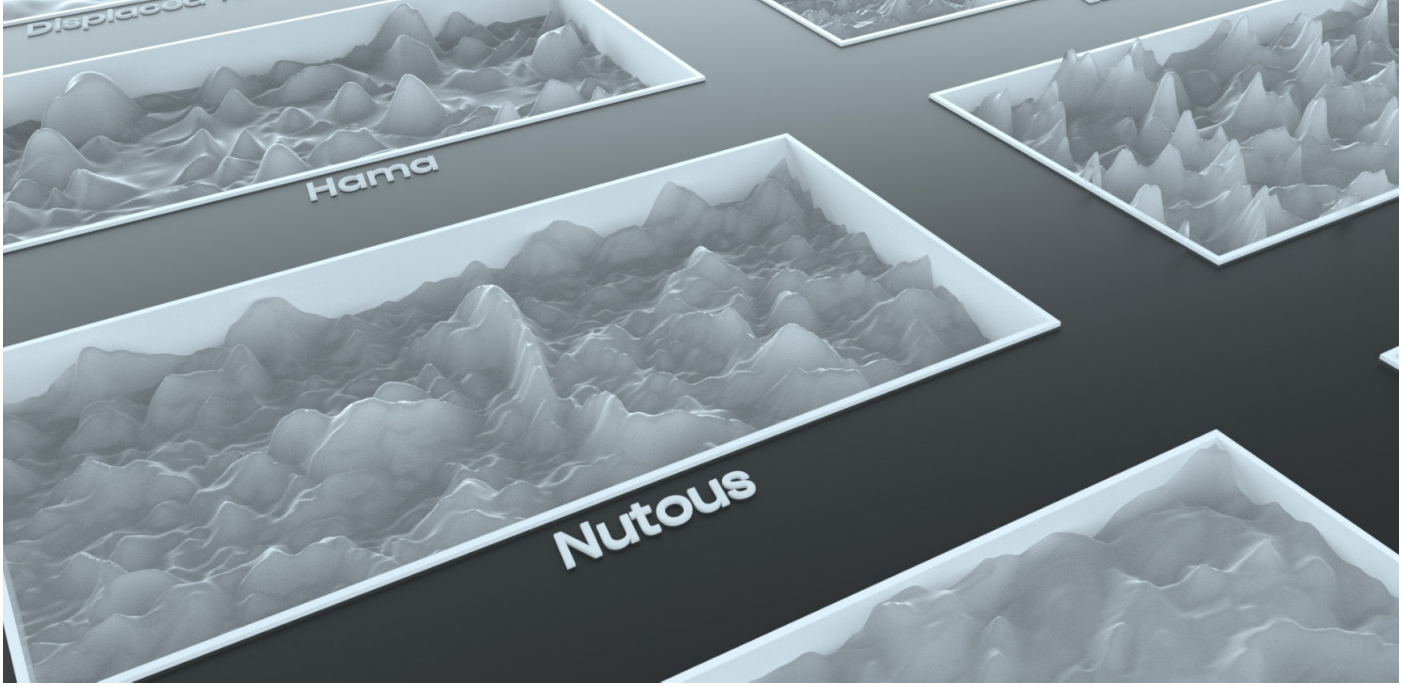


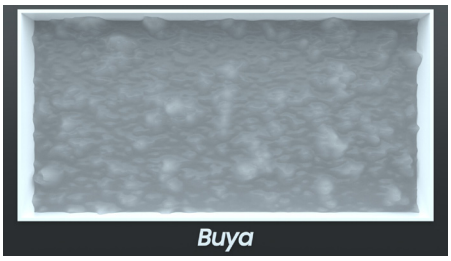


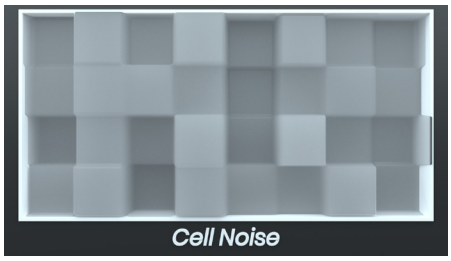


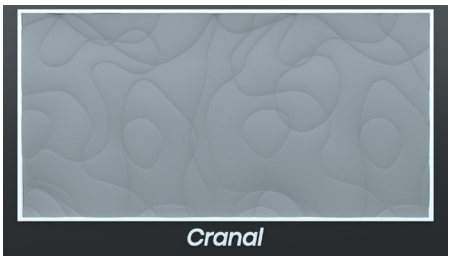


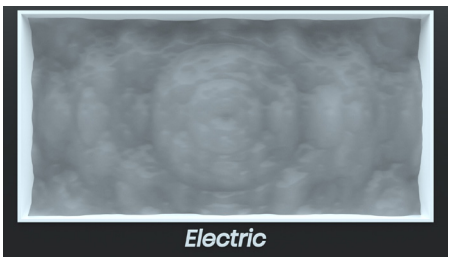


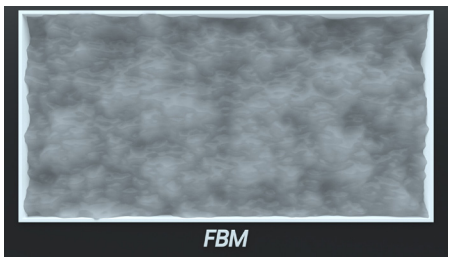


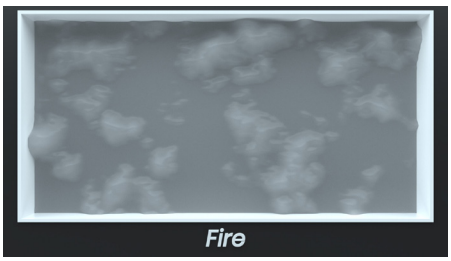

Buya

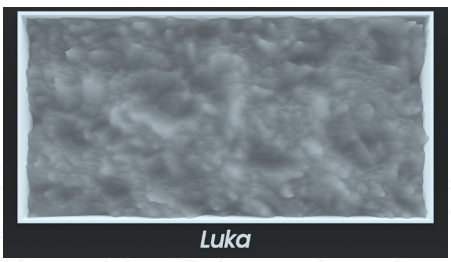

Cell Noise

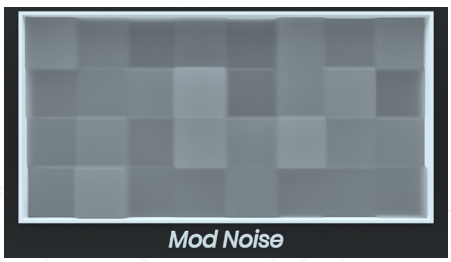

Cranal

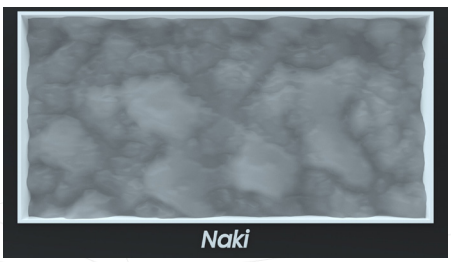

Electric

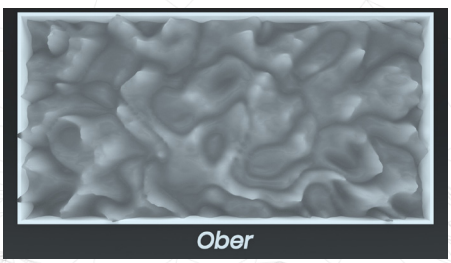

FBM

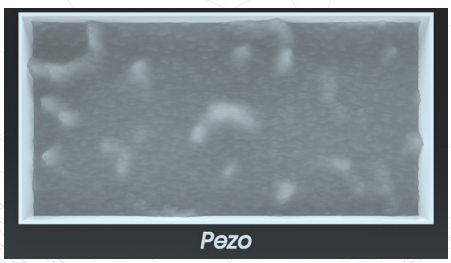

Fire

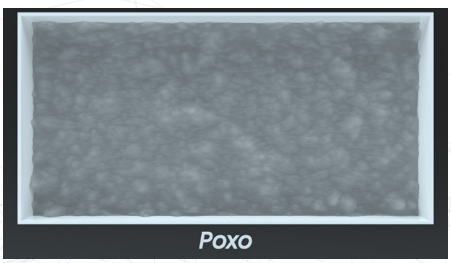

Luka


Mod Noise

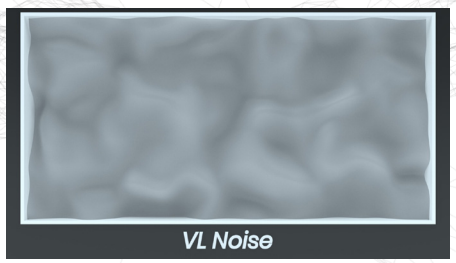

Naki

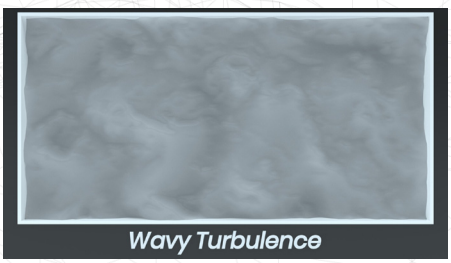

Ober


Pezo

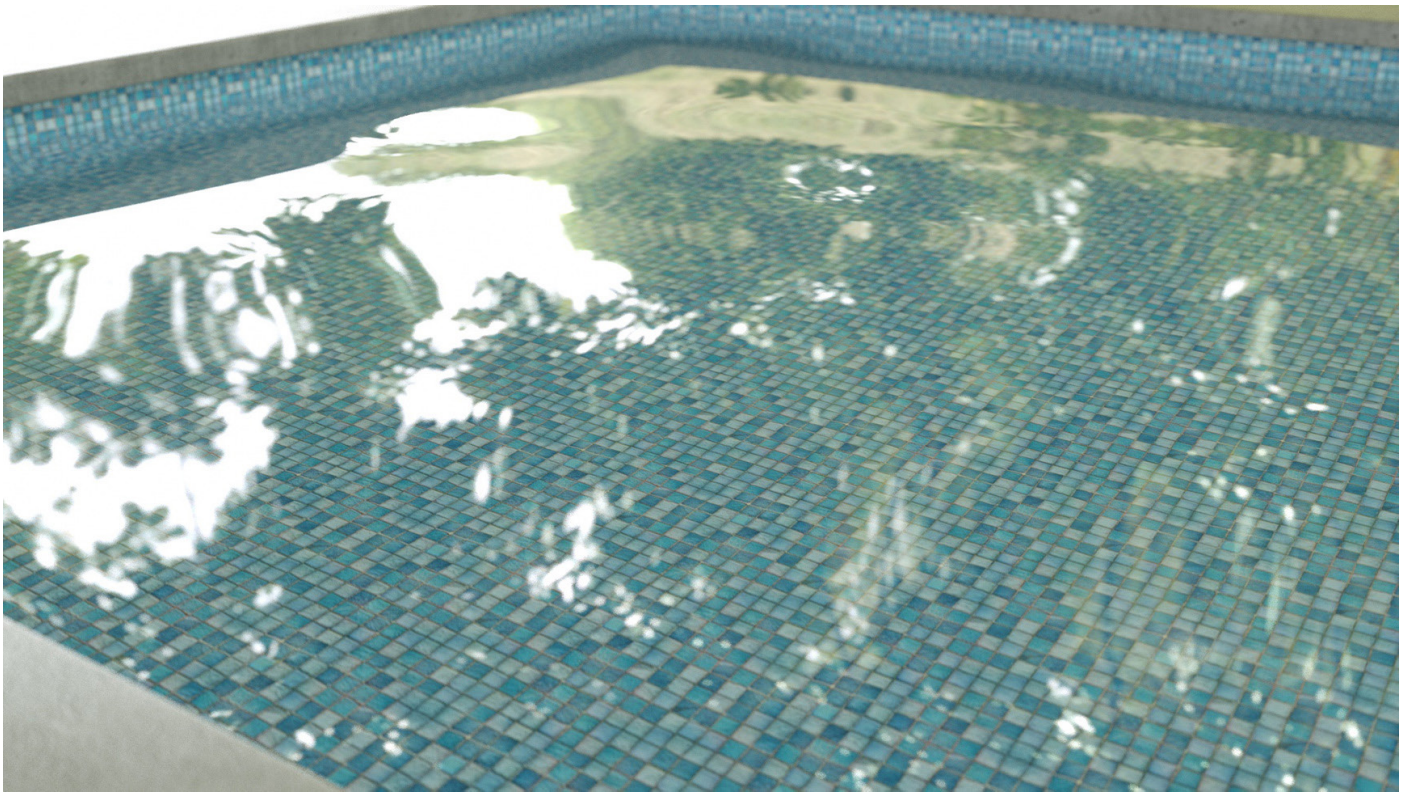

Poxo

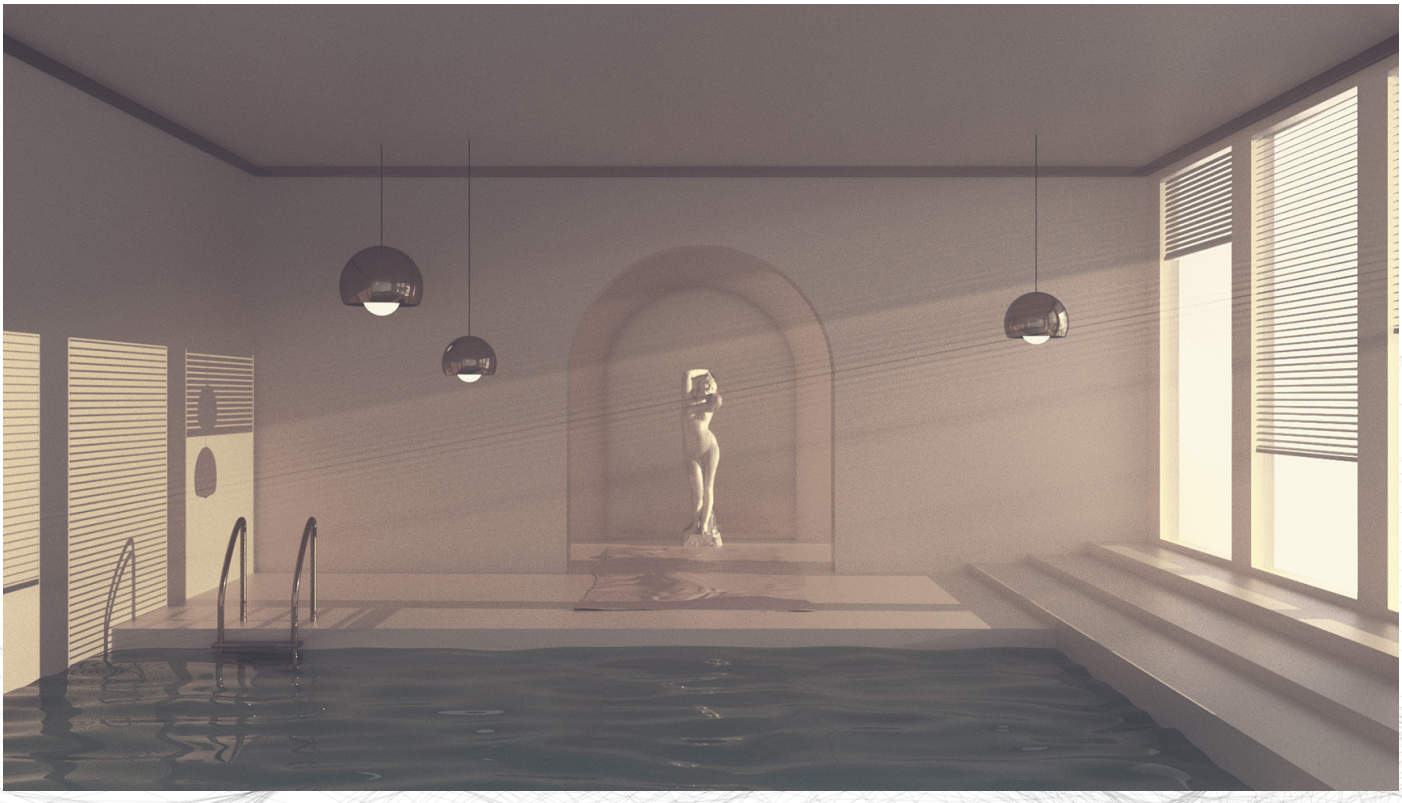

Turbulence


VL Noise

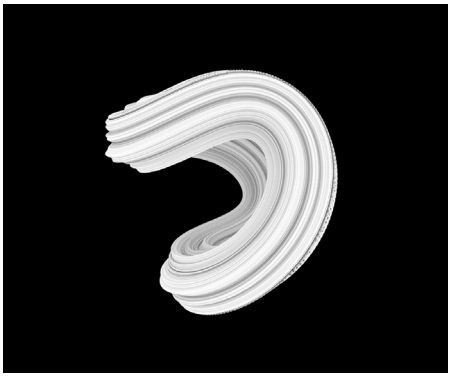

Wavy Turbulence

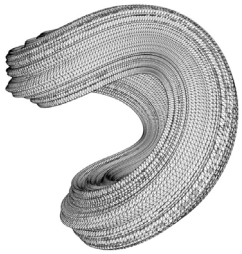
Rendu d’eau

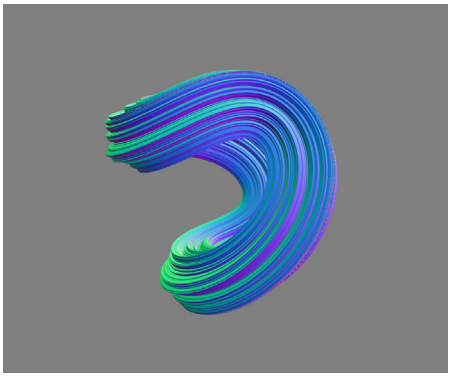


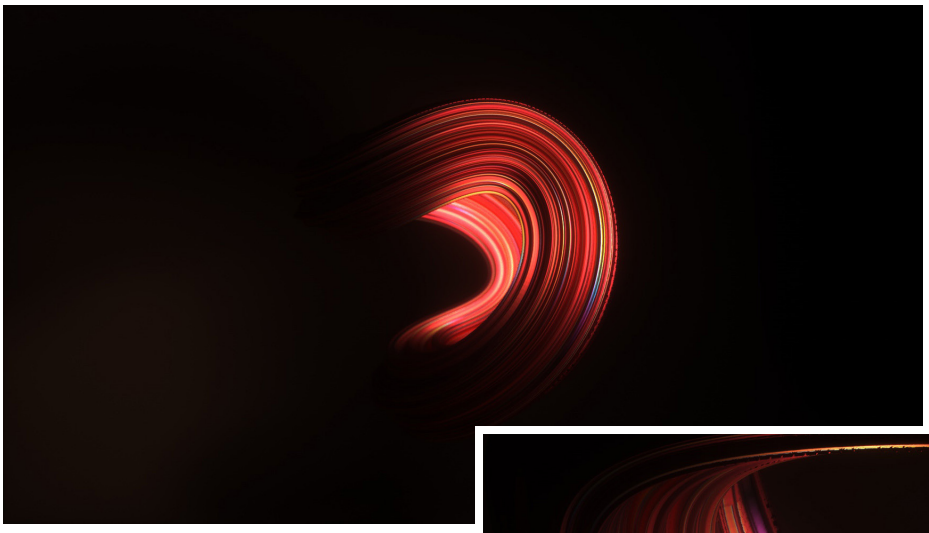


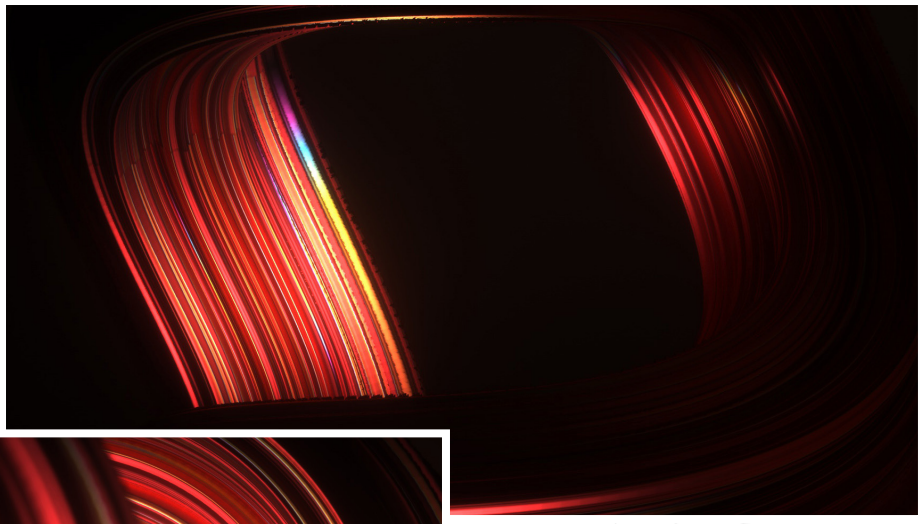
Échelle Relative

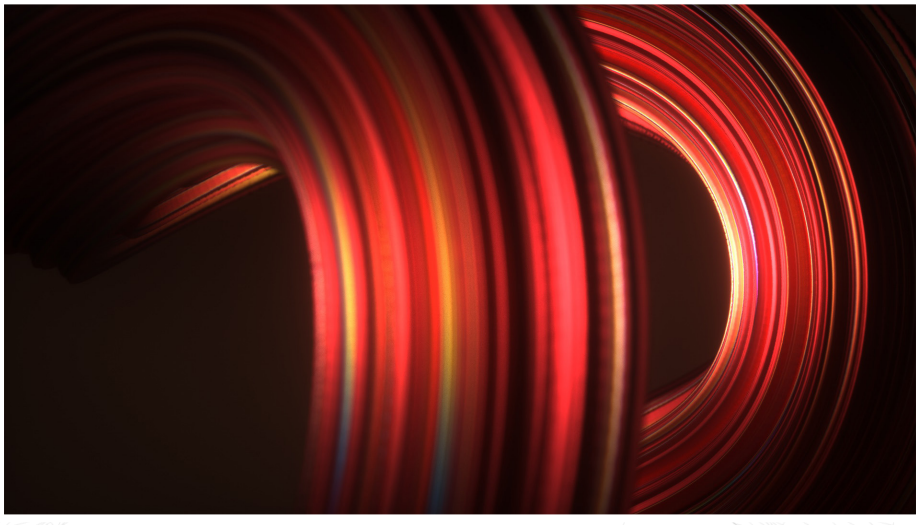




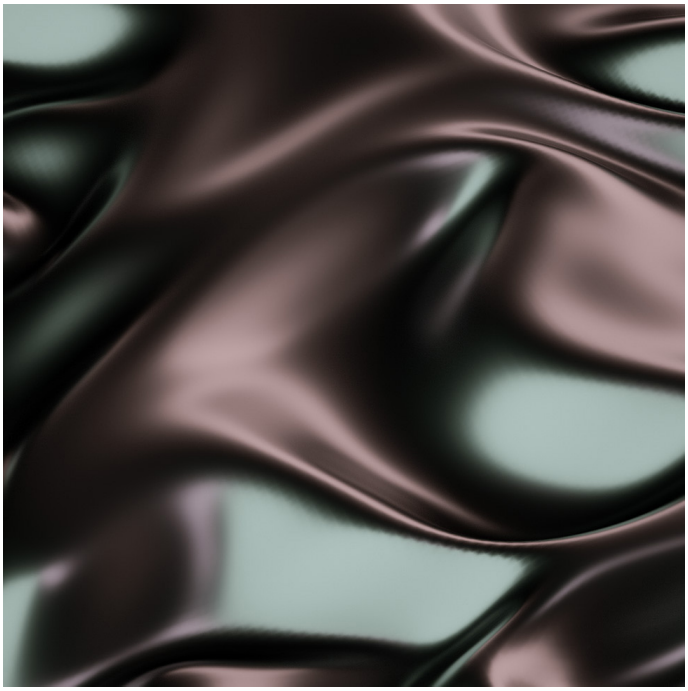


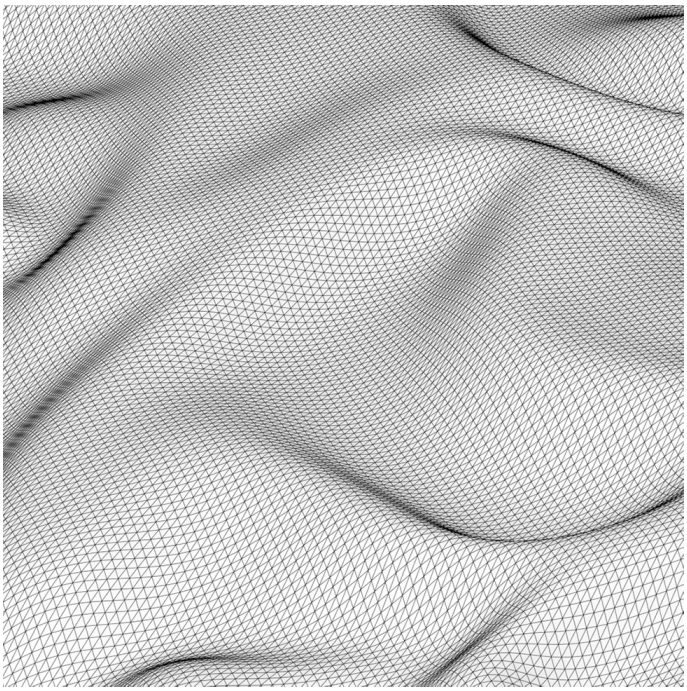






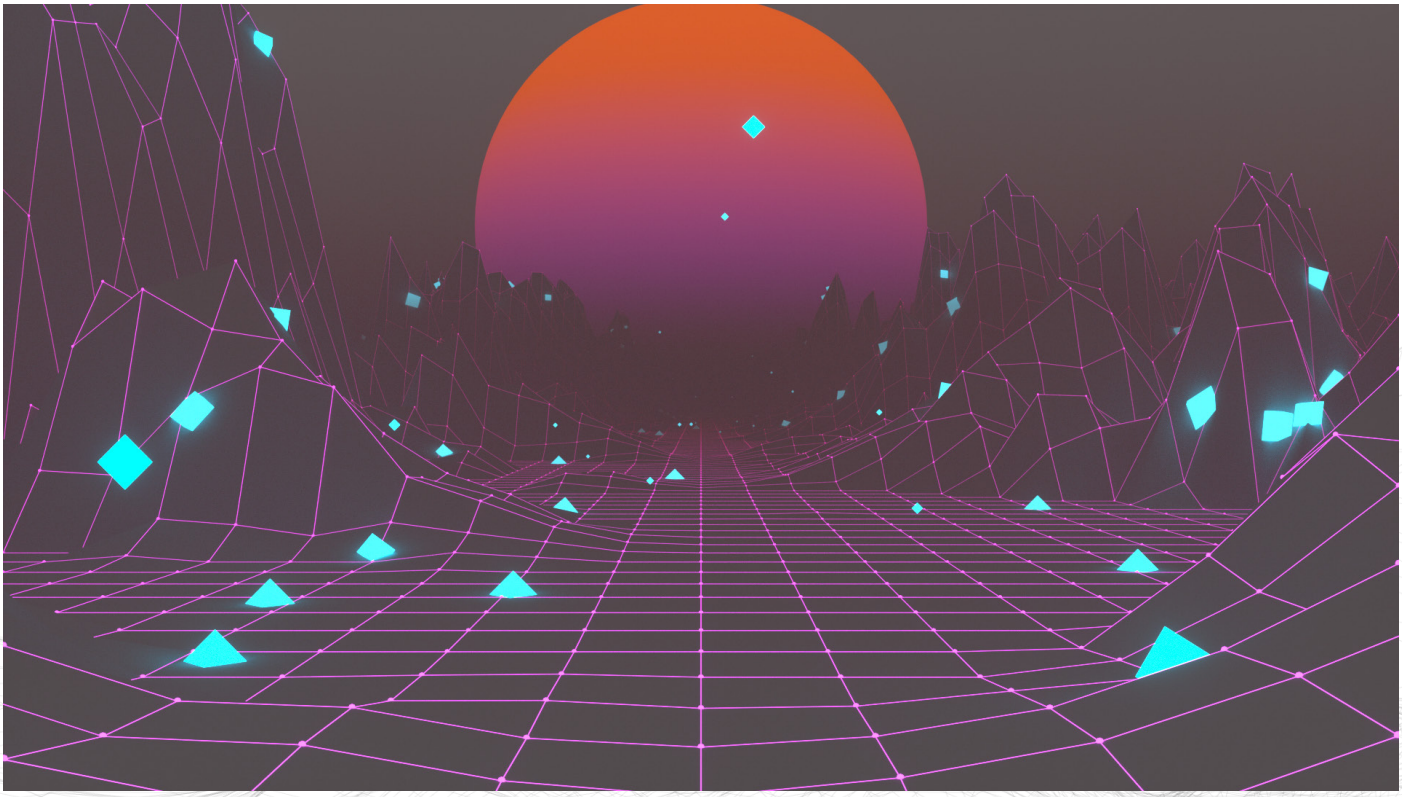
Rendu d’eau





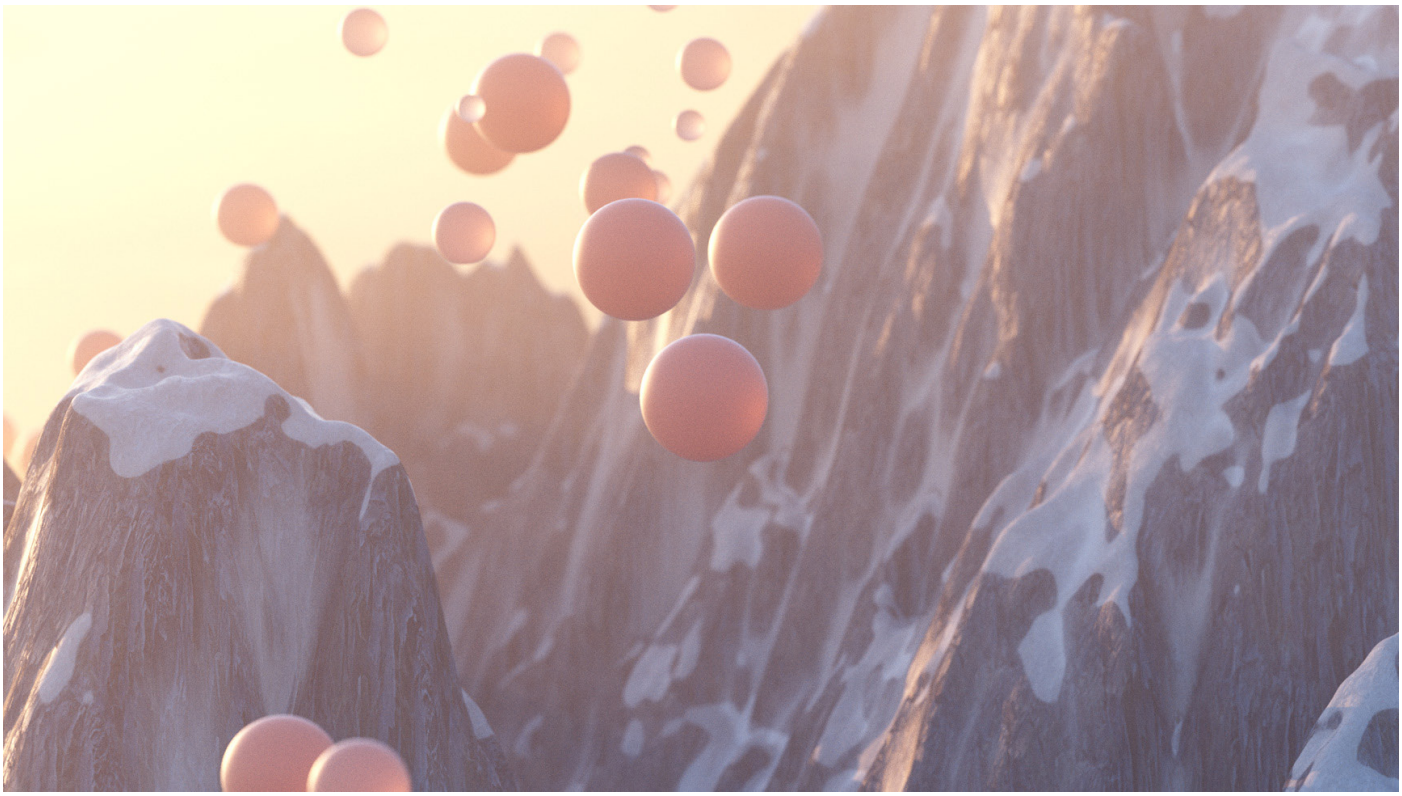
Animation Iridescence :
https://www.ludwigdejonckheere.com/noise/assets/iridescence/iridescent.mp4

Retro Terrain



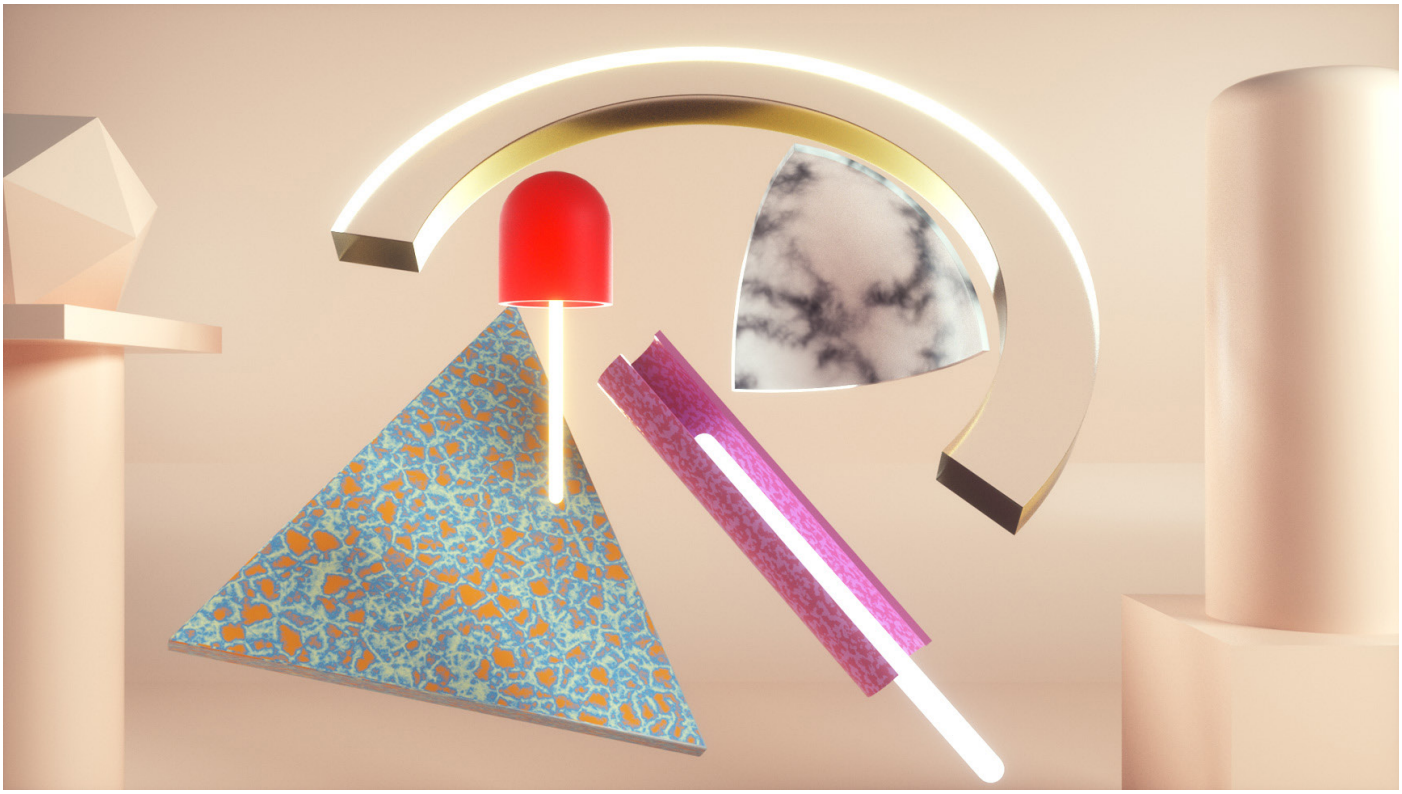
Animation Retro Terrain :
https://www.ludwigdejonckheere.com/noise/assets/retro.mp4

Montagne



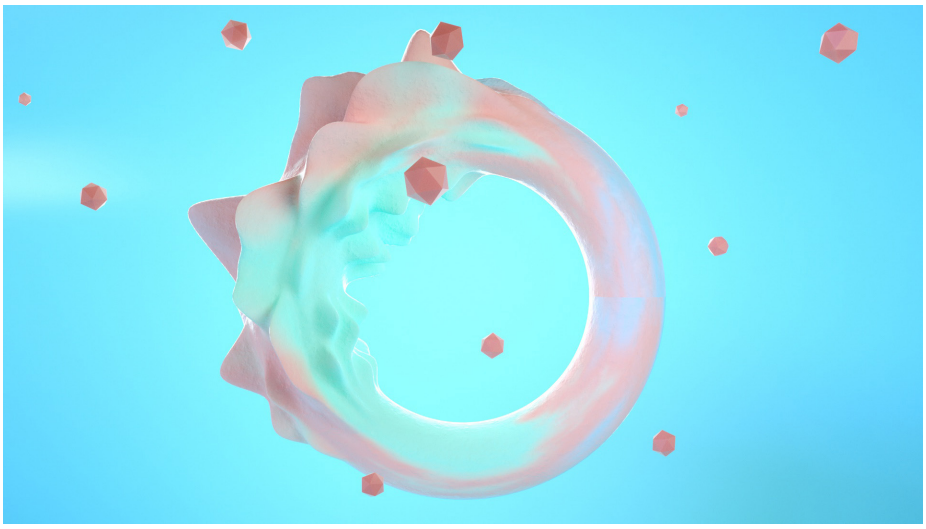
TEXTURING

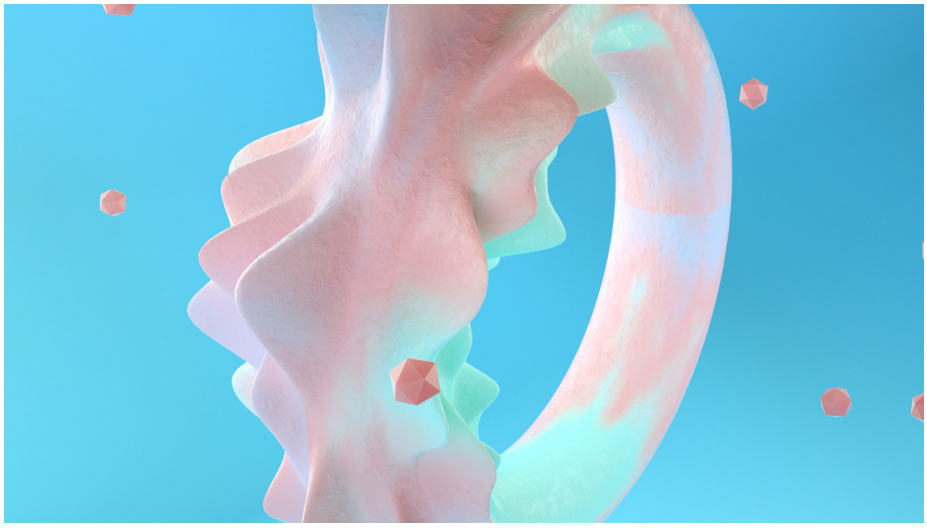
Diffuse

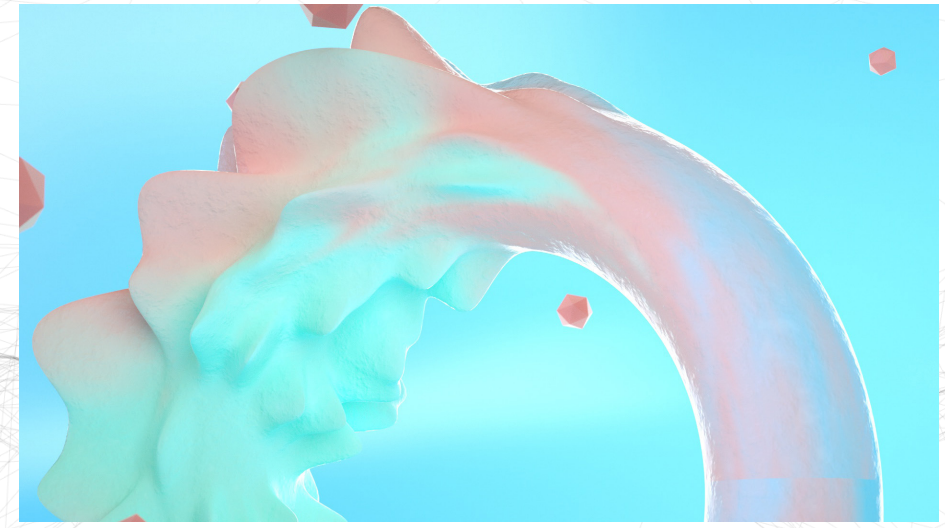


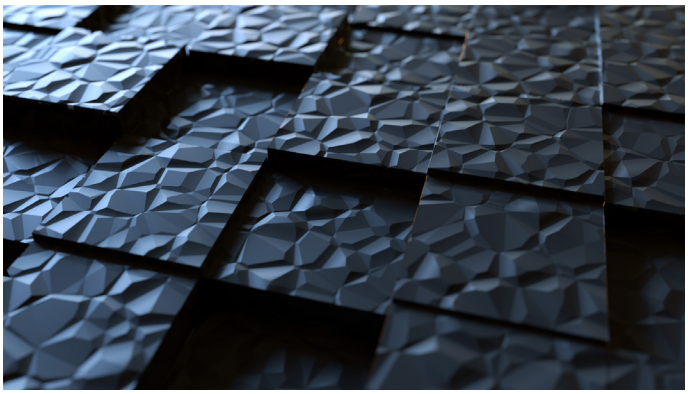
Roughness

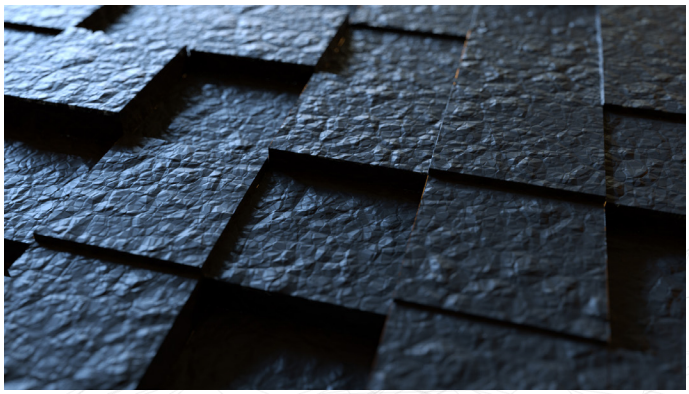
Bump

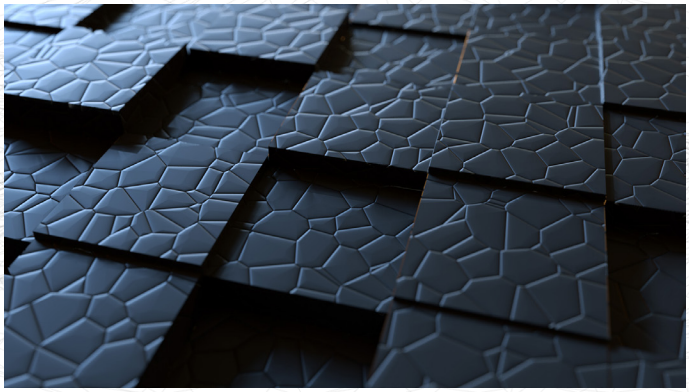


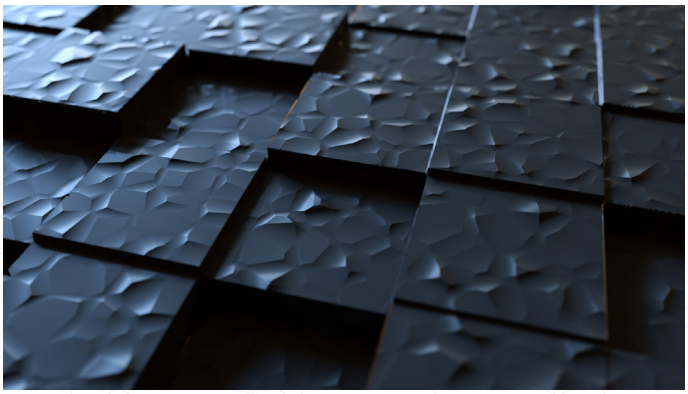


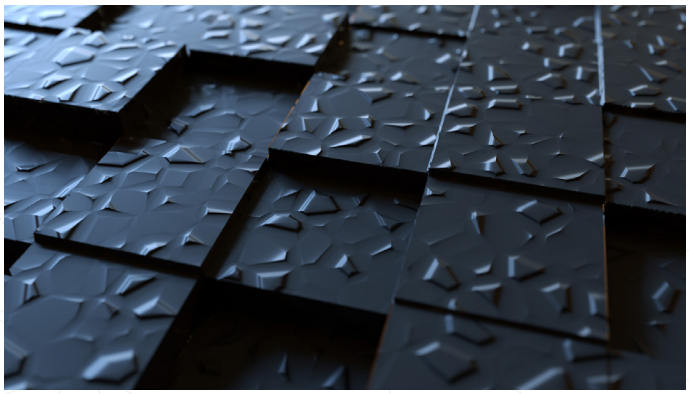


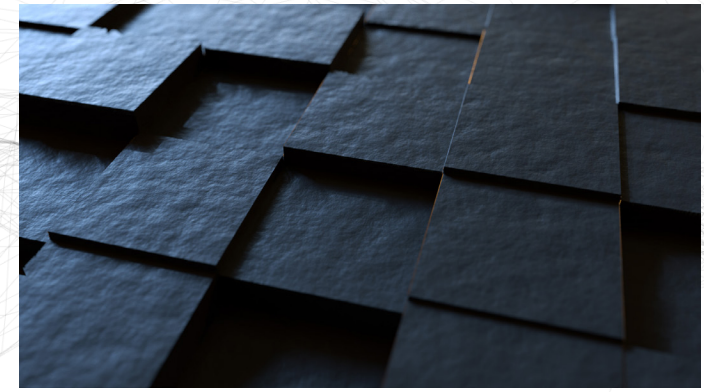


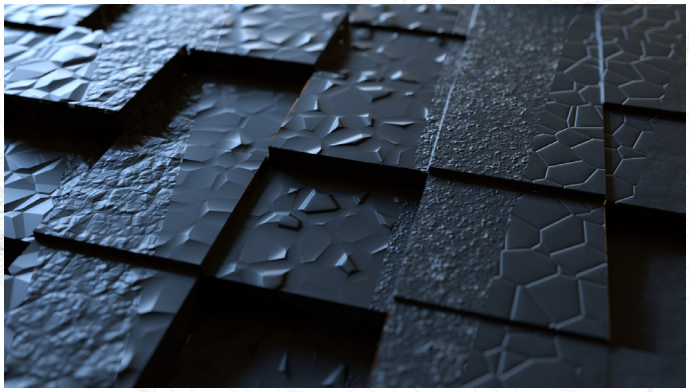




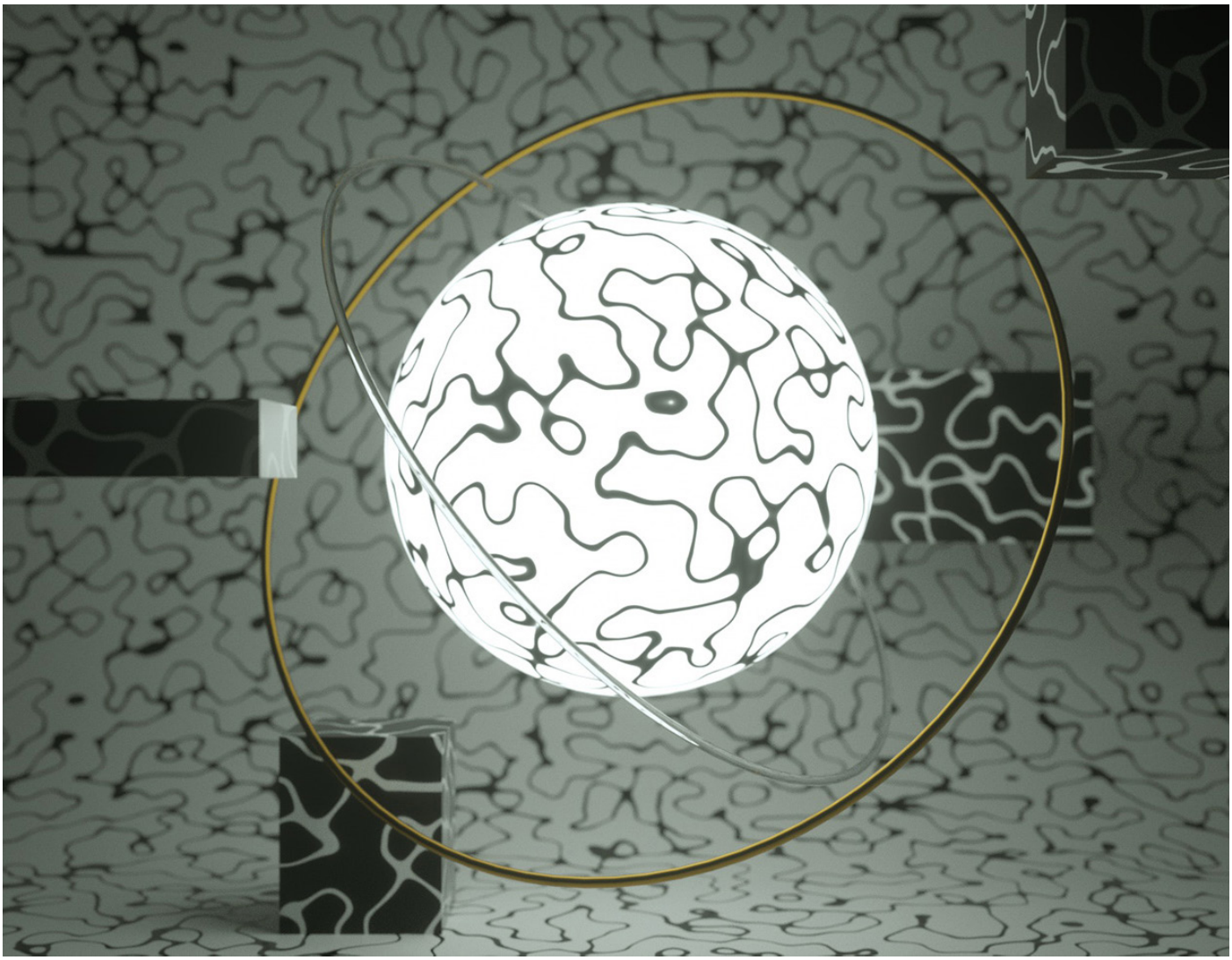




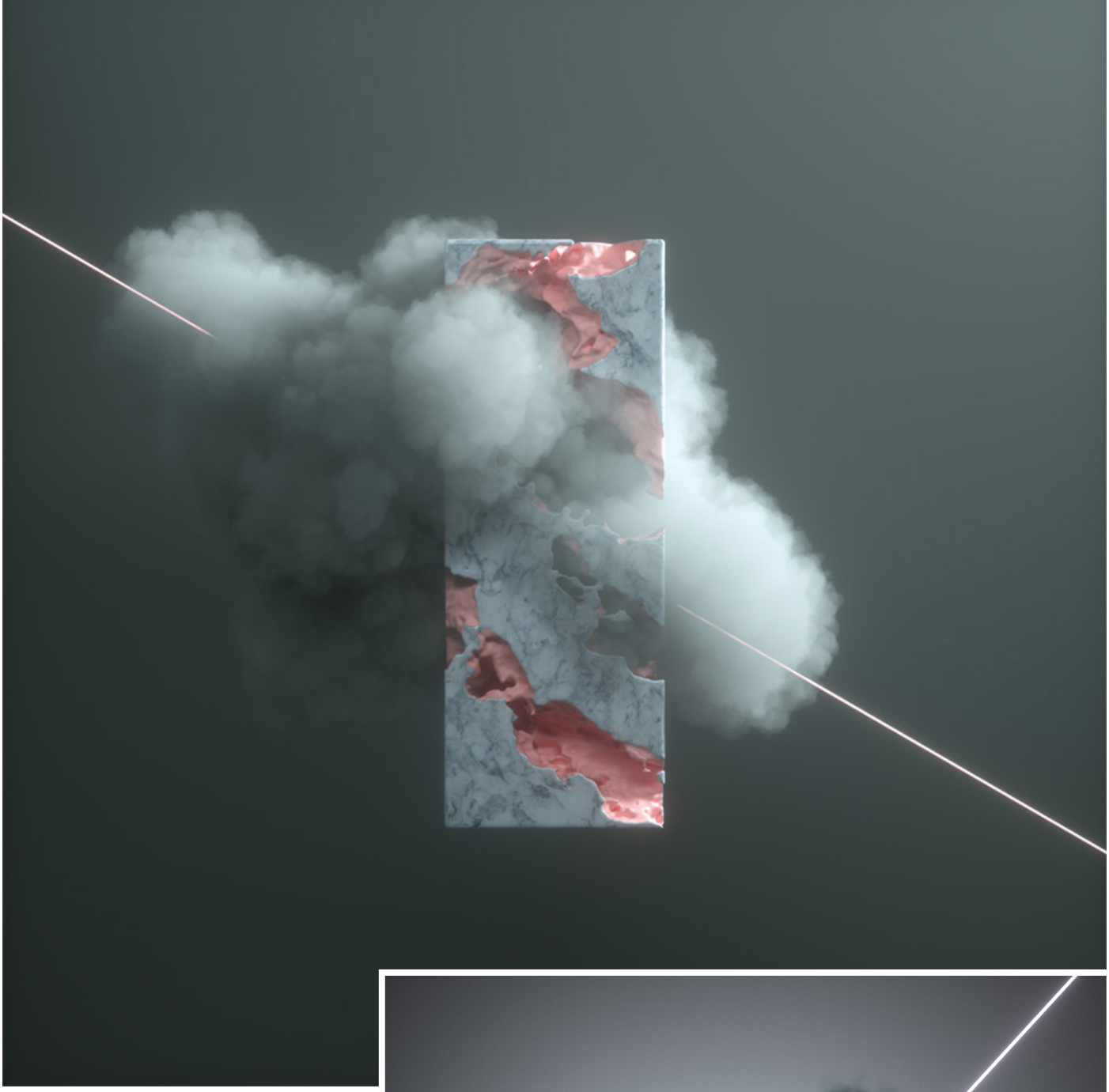


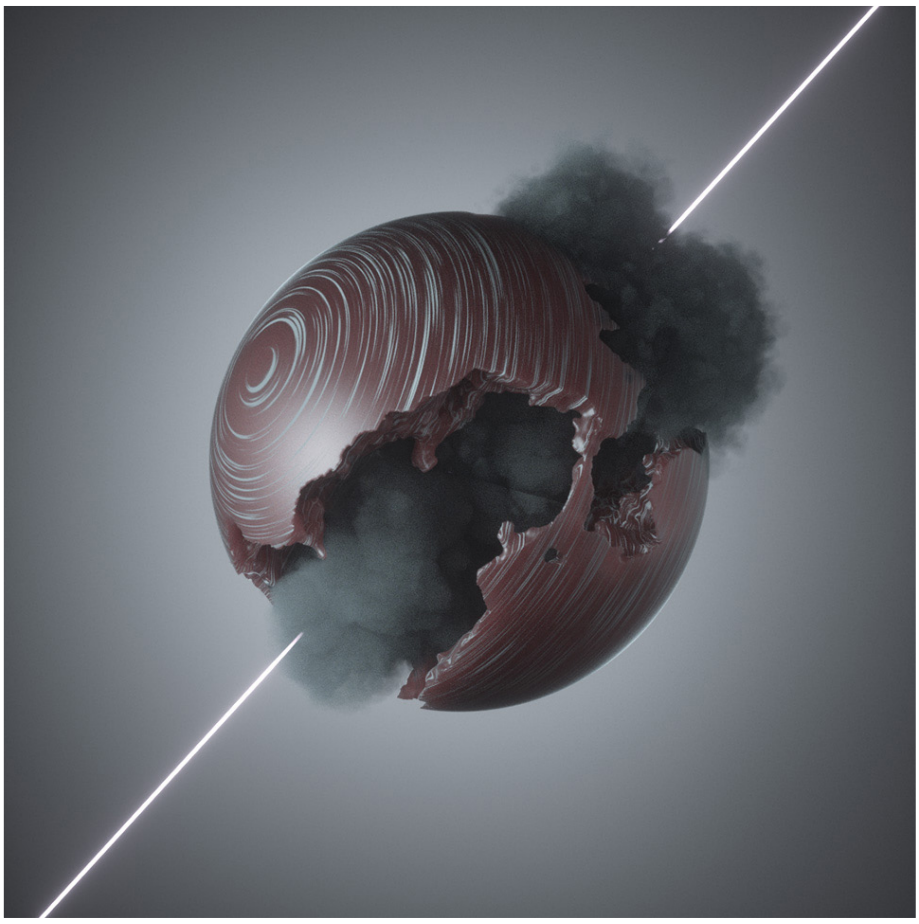


Emission



Animation Emission :
https://www.ludwigdejonckheere.com/noise/assets/emission.mp4

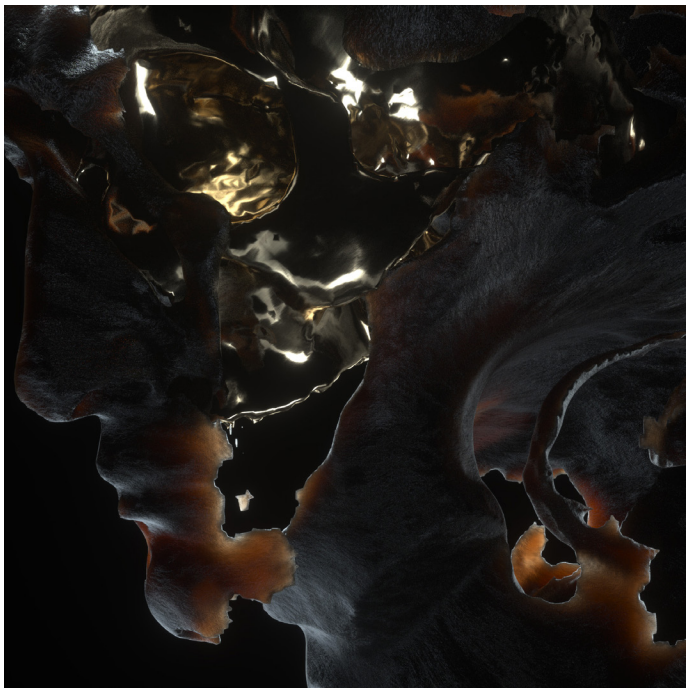


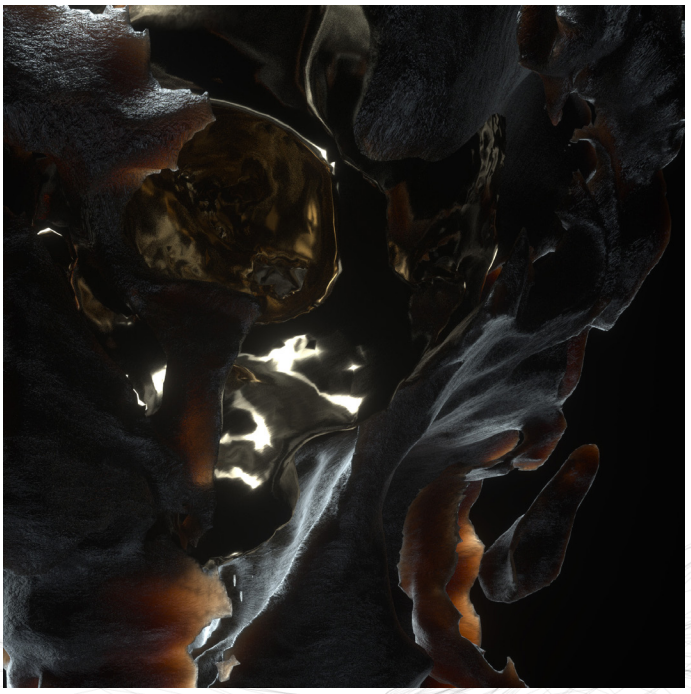


VOLUME

Marble & Ball

David



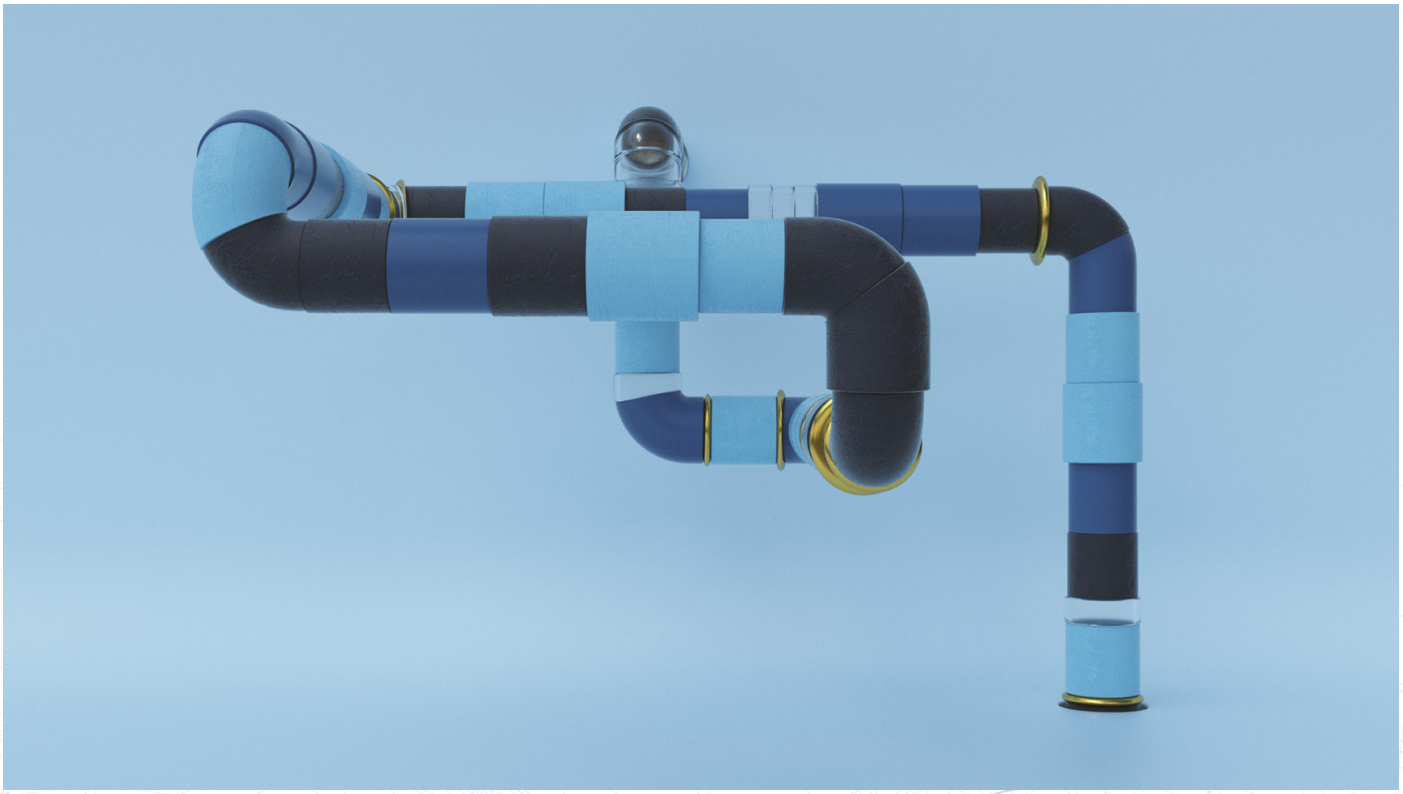


ANIMATION

HUD

Animation HUD :
https://www.ludwigdejonckheere.com/noise/assets/animation/tracking_v2.mp4

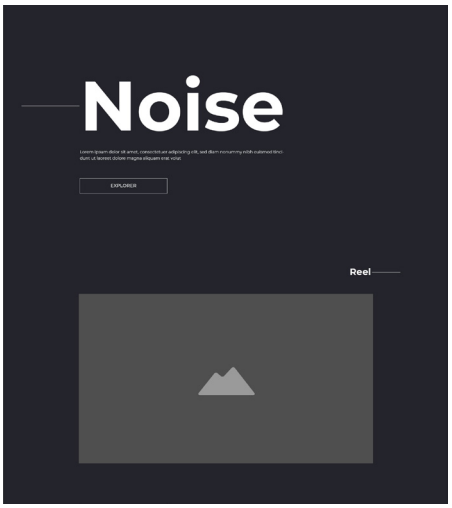
Tubes

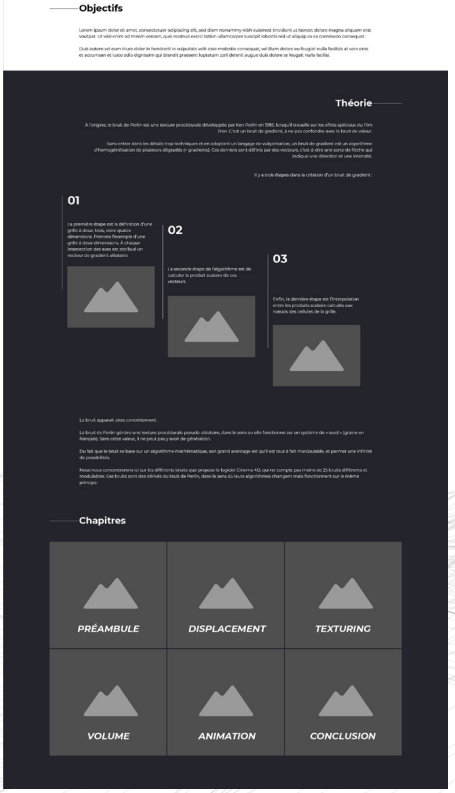


Animation Tubes:
https://www.ludwigdejonckheere.com/noise/assets/tubes_anim.mp4

SITE WEB

Wireframes





Texturing

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tinci
dunt ut laoreet dolore magna aliquam erat volut

Qu'est-ce que le texturing ?

Le « texturing » se réfère à la création de matériaux. Un matériau est constitué de plusieurs caractéristiques permettant de rendre
ouellement tous les matériaux existant dans le réel, qu'il s'agisse de verre, de plastique, etc.

Dans ce contexte, le texturing se fait via l'éditeur de matériaux d'Octane, le moteur de rendu, et non pas via l'éditeur de matériaux standards
de Cinema 4D.

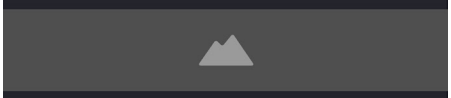
Parmi ces paramètres, il y a :

- « diffuse » : la couleur générale du matériau, que ce soit une couleur unie, un dégradé, un bruit, etc.
- « roughness » : la rugosité du matériau. Une faible rugosité rend la surface brillante, tandis qu'une forte rugosité la rend plus mat
- « bump » et « normal » : ces deux paramètres simulent un relief depuis une texture. L'apparence du matériau change et peut donner cette illusion alors que la géométrie ne change pas.
- « displacement » : paramètre relatif au chapitre précédent, mais il s'agit ici du matériau, et non pas de l'objet 3D. Cependant, comme pour le bump et la normale, la géométrie n'est pas modifiée.
- « emission » : tout simplement l'opacité du matériau.
- « emission » : le matériau devient une source lumineuse.

Chaque paramètre décrit ci-dessus peut se voir attribuer une texture, et on l'accurrence un bruit.

Ici on voit clairement le bruit en lui-même, il est spécifié, dans la plupart des cas, du bruit d'Octane, et non pas de Cinema 4D. C'est presque
similaire, mais les paramètres sont présentés autrement.

Il y a bien d'autres paramètres qui valent selon le matériau, mais ceux-ci sont les plus importants pour la suite.



Diffuse

En commençant par la caractéristique « diffuse » du matériau, le bruit
agit directement sur la couleur de la texture. Le bruit est
basiquement une nuance de noir et blanc, mais on lui associant un
dégradé d'Octane, le bruit présente alors des variantes de couleurs.
On dégradé contient au minimum deux couleurs, mais peut aussi en
présenter davantage.

Roughness

Le rugosité d'un matériau va drastiquement changer son
comportement vis à vis des reflets de lumière. Plus le matériau est
lisse, plus les reflets seront définis et nets. Si celui-ci est rugueux, les
reflets seront alors plus atténués et doux.

Ce paramètre peut aussi être associé avec des bruits, et ainsi de
retour le plus lisse, et à la valeur la plus rugueuse. Mais la rugosité peut
aussi être définie en fonction d'une texture, et donc un bruit. Les
nuances sombres et noires où ce bruit sont associées à l'aspect lisse,
et les nuances claires et blanches sont alors rugueuses.

Une bonne application du bruit dans le paramètre de rugosité est
frappante lorsqu'il s'agit de matériau spéculaire, comme du verre par
exemple.

Dans cet exemple, il est question d'une forme géométrique,
l'icosaèdre, auquel un matériau semblable à du verre est appliqué. Au
niveau de la rugosité est attaché une texture de bruit, qui donne alors
un aspect goté ou sali au verre.

Bump

Le paramètre de « bump »
simule à donner un relief
au même titre que les «
normal ». Cependant, le
paramètre de normal
prend en compte un
autre type de texture
concernant des valeurs
de couleurs selon les axes
X, Y et Z. Un bruit est donc
inutilisable pour ce type.
Mais ce n'est pas le cas du
« bump », qui prend en
compte les nuances noires
et blanches.

Ce paramètre est assez
similaire au « displacement
», à la différence que la
géométrie ici reste
inchangée.

Sur cet exemple de bruit,
le détail de la texture est
assez fin et est généré via
un bruit dans le « bump ».
Ceci rend alors le matériau
plus intéressant et riche.

C'est aussi une opportunité de rajouter du détail supplémentaire à plus petite échelle sur des surfaces qui présentent déjà un «
displacement ». En mixant alors ces deux paramètres, on obtient un terrain complexe et organique.

Emission

Enfin, le dernier paramètre de texture présenté est le canal « emission », ou émission en français. Il s'agit plus subjectivement de rendre une
surface lumineuse. Pareillement que les paramètres présentés précédemment, un bruit peut très bien être mixé en tant que source de
lumière.

Dans cet exemple, c'est le bruit noir et blanc qui est source de lumière, où la partie blanche devient alors émettrice.

Chapitre Précédent

Chapitre Suivant

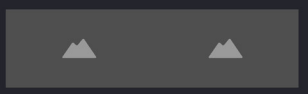
Volume

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tinci
dunt ut laoreet dolore magna aliquam erat volut

Que sont le Volume Builder et le Volume Mesher?

Ces deux fonctionnalités sont apparues dans la version R20 de Cinema 4D et fonctionnent ensemble.

Premièrement est appliqué le « volume builder », qui va/des objet(s) 3D, i.e « volume builder » transforme alors la géométrie en un ensemble
de « voxels », une sorte de pixels 3D. Ces derniers prennent l'apparence de petits cubes blancs. Le taille des pixels 3D peut varier, en étant
ainsi le résultat final plus ou moins détaillé. Si la taille des petits cubes est petite, le résultat est plus précis. À l'inverse si les « voxels » sont de
plus gros de cubes, le détail est perdu. Attention cependant, plus les pixels 3D sont petits, plus il faudra demander de ressources, et donc
talent, voire de fermer ou ne répond plus.



Ce « volume builder » va donc transformer une forme géométrique
en un ensemble de pixels 3D. Sur un objet 3D simple comme une
sphère ou un « Jet, l'utilité est nulle. Si maintenant plusieurs objets
complexes sont chevauchés et entremêlés, alors cette fonctionnalité
est intéressante.

En effet, les diverses formes géométriques vont fusionner et se
souder les unes aux autres, pour créer une forme globale.

Seulement, le petit truque, il n'est vraiment qu'un nuage de « voxels »
Cela donne la sensation que le « volume mesher » va par la transformer
ces cubes en une vraie forme géométrique composée de polygones.

Plusieurs modes sont proposés avec ces nouvelles fonctionnalités, un peu à l'image de l'outils « booléen », un outil très connu dans le monde
de la 3D.

Basiquement, ces modes sont au nombre de 3 :

01

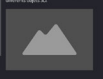
Union : permet d'unir plusieurs objets 3D
pour n'en former qu'un.

02

Soustraction : consiste la forme d'un objet
en fonction de la forme des autres objets.

03

Intersection : ne prend uniquement en
compte que la partie intersectée entre les
différents objets 3D.

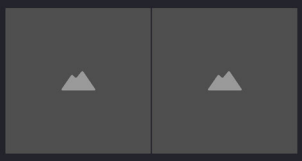


Dans le cadre du bruit, et concernant le « volume
builder », ce calcul n'inclut ni « fields » ni « cloner » ni
objet 3D gérée à un champ « aléatoire », qui est le bruit. En
utilisant le mode « exclusion », la forme est creusée par
le bruit paramétré auparavant.

Marble & Ball

Dans ces exemples, l'idée est d'utiliser ces formes simples pour ainsi creuser l'idée à l'aide du bruit. Le résultat donne alors quelque chose
d'organique et aléatoire, avec des cavités.

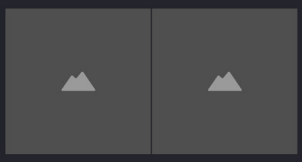
Pour donner l'illusion que le cœur de la forme est différent de sa surface, il faut sélectionner tous les polygones qui sont « à l'intérieur » de
cette dernière, et ensuite attribuer un « nouveau » matériau.



David

Dans cet exemple, il n'est plus question d'objets 3D
simples, mais d'appliquer cette méthode sur des
formes plus complexes. La démarche reste similaire,
mais il s'agit ici d'une statue scannée en 3D, la bibi de
David par Michel-Ange.

Lorsque le tête est creusée, l'intérieur est visible et laisse
alors apparaître un « dire en or.



Chapitre Précédent

Chapitre Suivant

Animation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tinci
dunt ut laoreet dolore magna aliquam erat volut

Animations Aléatoires

Pour ce dernier chapitre, le principe est d'animer des objets de manière aléatoire, sur base d'un bruit. Ce bruit n'est donc pas visible
concrètement comme les exemples précédents, mais agit plutôt comme un champ de valeurs quelconque qui affecte la position, la
rotation, l'échelle, etc.

HUD

Pour cet exemple, l'animation est décomposée en plusieurs parties, qui seront unies en un projet final regroupant tous ces éléments.

En premier lieu, il s'agit d'animer une
couche selon le bruit. N'ayant pas question
ici de « displacer », mais de « dhader », un
autre outil du MoGraph de Cinema 4D.

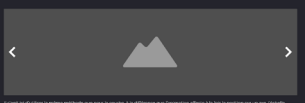
Dans les paramètres de ce « dhacke » est
explicqué en détail sur sera la source du
mouvement. Pour ce projet, le « Cell Noise
» sera approprié.

Dans à déterminer ce qu'affecte le bruit.
Dans le cas de cette couche, il s'agit de la
« shader » aux points de la couche, et non
pas à l'objet.

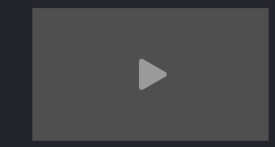
Pour que la déformation prenne effet, il
s'agit donc un second temps d'affecter le «
points de la couche, et non

C'est en définissant enfin une vitesse au bruit que l'animation
se créer. Le « Cell Noise » a cette particularité d'avoir une
animation très dérivable, ce qui est recherché dans ce projet.

Vient ensuite l'animation d'ainsi de cercle ainsi que de la courbe précédente. À cette fin, ces différents éléments sont entreposés dans un «
cloner ». C'est sur ce dernier qu'un nouveau « shader » prend effet.



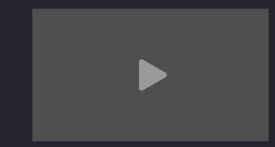
Il s'agit ici d'utiliser la même méthode que pour la courbe, à la différence que l'animation affecte à la fois la position sur un axe, l'échelle
globale, ainsi que la rotation selon un axe.



Tubes

Dans cet exemple, la méthode reste la même, à savoir
l'utilisation du « shader » sur un « cloner ». À la
différence que dans ce projet, il s'agit uniquement
d'affecter l'échelle de chaque élément.

Ce projet consiste en un tube duquel sort de la « crème »,
dans le « shader » effacé Rochelle, grâce à un bruit, ici
le « Caseux ». Chaque tube voit alors son échelle
varier et donne alors naissance à une armada en
silestaire.



Chapitre Précédent

Chapitre Suivant

Que signifie « le bruit en infographie » ?

Ce travail met en lumière les différents usages du bruit de Perlin lorsque l'on traite de créations 3D.

Qu'est-ce que le bruit de Perlin ? Quels côtés pratiques présente-t-il concernant le monde de a 3D ?

What does « noise in infographics » means ?

This work sheds lights on different purposes of the Perlin noise when dealing with 3D creations.

What practical sides does it present concerning the world of 3D?