

UNIVERSITÉ PARIS 8

**Spécialité de Master Arts et Technologies de l'Image Virtuelle
de la mention Arts Plastiques et Art Contemporain**

Contrôle de simulation physique
au service de l'artiste pour la création d'effets spéciaux.

Kévin Simorre

RÉSUMÉ :

Contrôle de simulation physique au service de l'artiste pour la création d'effets spéciaux.

Ce mémoire est un condensé des recherches accomplies tout au long de l'année dans le domaine des VFX, elles se centrent autour d'une problématique commune, qui est la création d'outils permettant de contrôler de manière précise une simulation physique pour les effets spéciaux. Pour ce faire, je me suis essentiellement servi du logiciel Houdini en raison de son système nodal qui offre la possibilité de maîtriser en profondeur l'ensemble des outils qu'il met à disposition.

ABSTRACT :

Physical simulation control in VFX for the artists.

This Master Thesis is a summary of all the investigations I have carried out throughout my year in the field of VFX. They are focused around one problematic : the creation of tools which allow to control in a very efficient way a physical simulation for visual effects. To do so, I primarily used the Houdini software which offers the possibility to control in-depth all the tools that are available thanks to its nodal system.

Merci à l'ensemble des professeurs et intervenants d'ATI.

Merci également à mon entourage, qui a su me ménager pour que ces années d'études soient effectuées dans de bonnes conditions.

À ma belle, mes parents, ma jumelle, ma famille et mes amis.

TABLE DES MATIÈRES :

INTRODUCTION :	6
LE FONCTIONNEMENT DU LOGICIEL HOUDINI :	8
1 : LA FUMÉE :	10
1.1 Contrôler la fumée :	10
1.1.1 Contrôler par des phénomènes physiques :	10
1.1.2 Avec des géométries 3D :	12
1.1.2.1 L'interaction :	12
1.1.2.2 Les normales :	13
1.1.3 Avec des particules :	14
1.1.3.1 Les volumes :	14
1.1.3.2 L'advection :	16
1.2 Rendre une fumée :	17
2 : LES PARTICULES :	18
2.1 Diriger les particules :	19
2.2 Rendre les particules :	20
2.2.1 Le partitioning :	20
2.2.2 Les passes :	21
3 : LA DESTRUCTION :	25
3.1 Briser le cocon :	25
3.1.1 Préfracturer :	25
3.1.1.1 Contrôler la taille des débris :	27
3.1.1.2 Optimiser :	28
3.1.2 Simuler :	30
3.1.2.1 Les solvers :	30
3.1.2.2 Créer des contraintes entre les débris :	32
3.1.3 Rendre un plan de destruction :	33

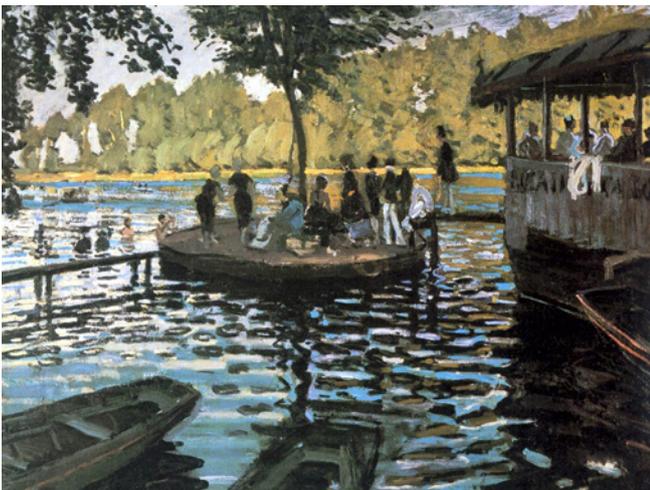
3.1.3.1	Éclairer un plan de fracture	35
3.1.3.2	Texturer les fractures	35
3.1.3.3	Ajout de détail	36
3.2	Dégrader une membrane :	37
3.2.1	Dans Houdini :	37
3.2.2	Dans Maya :	39
CONCLUSION :		40
BIBLIOGRAPHIE :		41
ANNEXE : LES VFX ET LA MACHINE :		43
ANNEXE : TUTORIAUX :		44
	Peindre une zone de chaleur sur un mesh - Le node Paint :	45
	Peindre une zone de chaleur sur un mesh - L'Attribut Transfer :	54
	Diriger des particules le long d'une curve - Point Cloud :	57
	Préfracturer & simuler - Glue Constraint :	70
	Répandre une couleur sur la surface d'une géométrie - Point Cloud :	88

INTRODUCTION :

Ce mémoire est indirectement issu de mon intérêt pour la peinture, et notamment celle de Turner et du mouvement impressionniste, qui tout au long de mes études artistiques m'ont rendu admiratif face à leurs manières de traiter l'eau et les manifestations physiques de la nature. C'est ainsi que des années durant j'ai tenté de les imiter.

Plus tard, lorsque j'ai entrepris mon apprentissage dans le domaine de la 3D, les effets spéciaux (vfx) m'ont permis de poursuivre ces recherches en m'offrant la possibilité de simuler de l'eau, du feu et des effets atmosphériques. Mais le manque de contrôle de certains logiciels me brida très vite. C'est pourquoi, disposant d'une année de recherche, j'ai décidé de vouer mon mémoire à la création d'outils permettant un contrôle précis d'une simulation pour les VFX, afin d'être en mesure d'obtenir une maîtrise artistique totale sur les effets que je souhaitais mettre en place.

Étant davantage créatif que développeur j'ai opté pour l'utilisation du logiciel Houdini pour effectuer mes recherches, il me donnait alors l'opportunité de créer mes propres " outils " sans avoir à écrire des lignes de codes complexes, ce qui pour autant ne m'interdisait pas de faire intervenir d'autres programmes dans mon processus de création.



Bain à la Grenouillère, Claude Monet, 1869, huile sur toile, 73 x 98 cm, Metropolitan Museum of Art, New York



Fisherman at Sea, William Turner, 1796, huile sur toile, 91 x 122 cm, Tate Gallery, Londres

Déroutement de mes recherches :

Cette année de Master s'est organisée en deux parties, la première de septembre à fin décembre visait à effectuer un travail de préproduction, pour un film réalisé en équipe sur une période de trois semaines. Dans ce court métrage en images de synthèse, il était question de nombreux types d'effets : créer une fumée qui glisse sur le sol et se transforme en particules, contrôler un flux de particules qui se déplace dans la végétation, et briser à la manière d'une pierre calcaire la surface d'un cocon.

Initialement, nous voulions mettre en scène l'éclosion d'un cocon au sein d'une forêt brumeuse. En dépit des décors qui étaient traités de façon réaliste, les effets spéciaux quant à eux venaient offrir une touche d'onirisme au film. Ce projet représentait pour moi une superbe occasion de m'exprimer, et j'espérais en ressortir des plans intéressants. Or la réalité était tout autre, car bien que la création des décors et des animations ait débuté plusieurs semaines en avance, je n'ai eu accès à des versions exploitables des scènes que trop tard, ce qui me laissait très peu de temps pour appliquer mes effets au film. Cependant, loin de me décourager, ce demi-échec au contraire me motiva à approfondir ma technique pour être encore plus efficace. Ce que je fis durant la seconde partie de l'année de février à mai.

Ce mémoire regroupe de manière chronologique les recherches effectuées au cours des mois pour le projet de court métrage. L'annexe donne accès à des tutoriaux rédigés dans le but de faire partager mes connaissances et d'apporter une meilleure compréhension du fonctionnement des effets que j'ai mis en place.

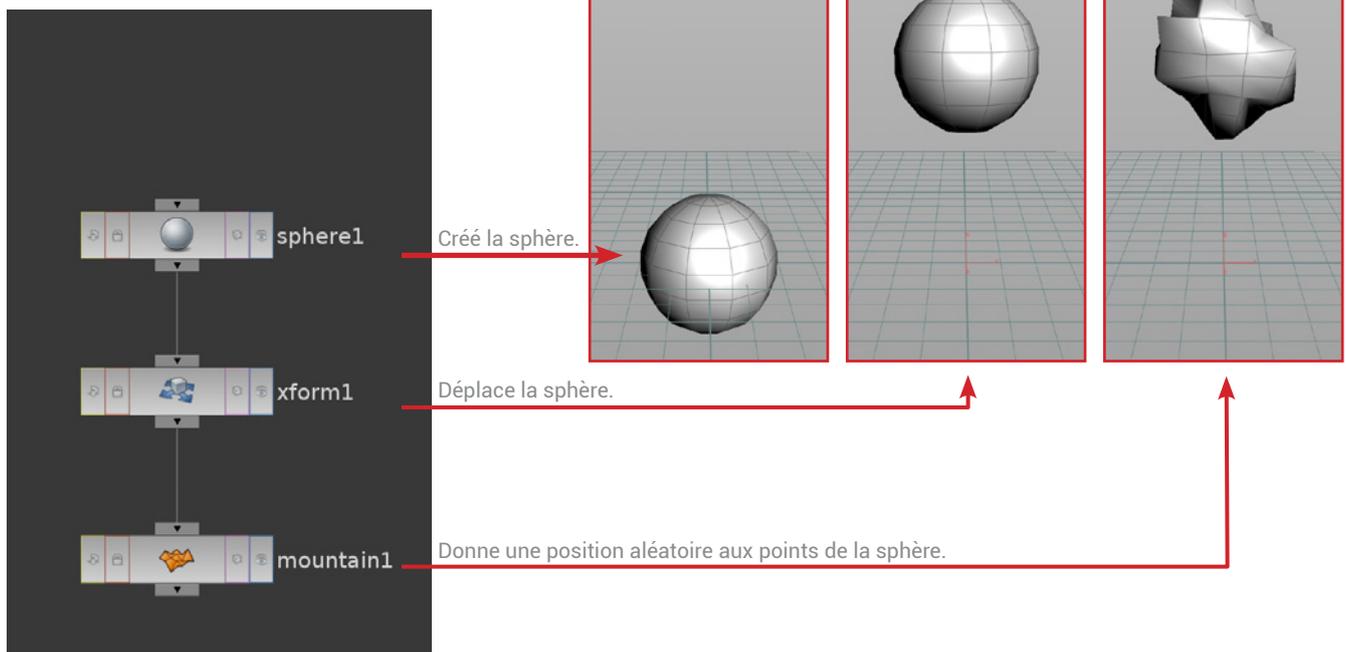
LE FONCTIONNEMENT DU LOGICIEL HOUDINI :

Voici quelques prérequis qui vous permettront de mieux comprendre ce mémoire.

Houdini est un programme à interface nodale. C'est-à-dire que chaque instruction donnée au logiciel est représentée sous forme de noeud (node) relié à d'autres par le biais d'entrées et de sortie. Chaque node à une fonction et des paramètres précis. Les entrées permettent au node d'obtenir des informations en provenance de la sortie d'un autre.

Dans l'exemple qui suit, nous créons, déplaçons, puis modifions une sphère :

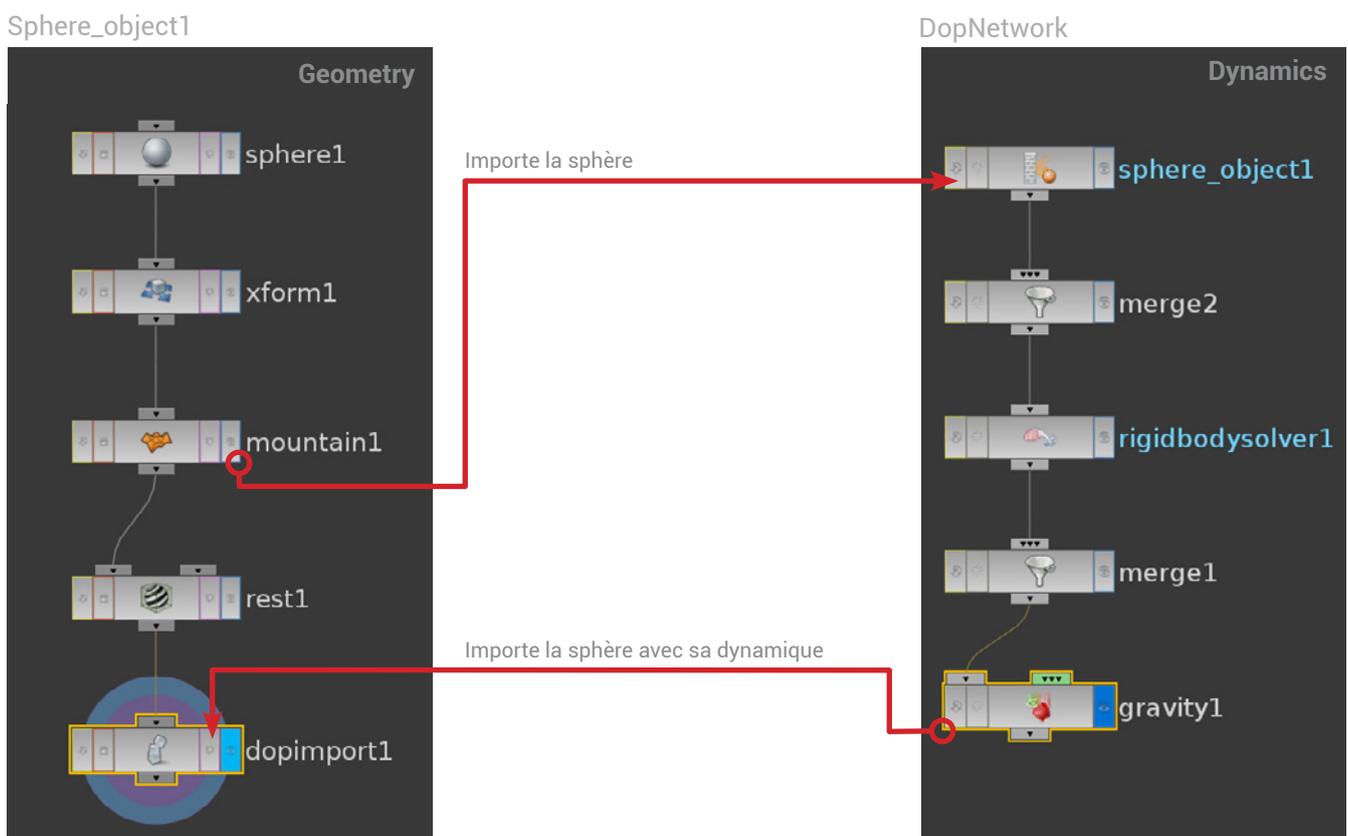
Les nodes :



Les paramètres des nodes :



La zone dans laquelle nous effectuons les connexions d'un node à l'autre est organisée en différentes couches (Scène, Géométrie, Dynamique, Texture, etc.). Chaque couche a une fonction spécifique, et peut communiquer avec les autres. Dans l'exemple précédent, nous avons modifié la forme et l'emplacement de la sphère dans celle nommée **Géométrie** ; dans le cas où nous voudrions rendre le mesh dynamique en lui donnant une gravité et des collisions il nous faudrait l'envoyer depuis la couche Géométrie (SOP) vers celle des Dynamics (DOP). Car seule cette dernière est capable de mettre en place une simulation sur un objet, alors que le niveau Géométrie a en charge tout ce qui se réfère au changement de forme.



Cette approche nodale à l'avantage d'être non destructive, car chaque paramètre peut être modifié à tout moment sans perturber le fonctionnement des autres nodes. Ainsi si je souhaite appliquer cet effet à une modélisation plus complexe, il me suffit de remplacer le noeud sphère par une nouvelle géométrie. Les nodes permettent de visualiser de manière logique le fonctionnement d'un effet, ce qui est utile lorsqu'il s'agit de reprendre un travail effectué plusieurs mois auparavant.

1.1 CONTRÔLER LA FUMÉE :

Maîtriser la dynamique d'un fluide de fumée est essentiel dans les VFX, car elle est présente dans la plupart des productions sous différentes formes telles que : la poussière, la vapeur, ou bien le gaz.

Le comportement d'un fluide est guidé par les lois de la physique, il est donc difficilement contrôlable. Dans mes recherches, mon but était de donner au fluide un itinéraire à suivre, sans le brider, afin qu'il garde une allure naturelle.

Ce chapitre aborde trois méthodes : le contrôle par des phénomènes physiques, par des géométries et par des particules.

1.1.1 Contrôler par des phénomènes physiques :

Pour les besoins du projet, mon premier défi consistait à créer une nappe de brouillard qui glissait sur le sol puis s'élevait avant de se transformer en particules.

L'approche initiale résidait dans le fait de peindre des zones de chaleur sur le sol, afin qu'à leurs contacts la fumée se réchauffe et devienne plus légère. Je souhaitais alors utiliser de véritables phénomènes physiques pour interagir avec le fluide. Bien que le principe soit simple, il me fallut du temps pour le concrétiser.

J'en étais alors au début de mon apprentissage sur Houdini, et j'ai dû faire face à de nombreux problèmes. Houdini 12 venait de sortir, or l'outil de gestion de fumée avait été entièrement remanié entre cette version et la précédente, j'ai donc eu des difficultés à trouver de la documentation à son sujet. Mais à force de tâtonnement, et en extrapolant avec les informations que j'avais recueillies sur son utilisation dans la version précédente, j'ai fini par parvenir à un résultat convainquant.

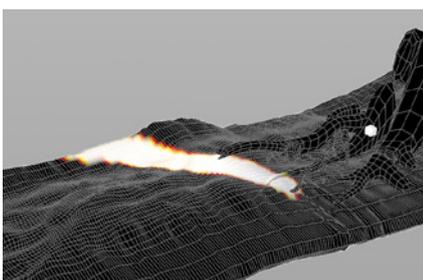
La première étape pour créer cet effet consistait à appréhender les différents

paramètres du solver de fumée, et ce n'est qu'une fois l'attribut permettant de gérer la température repéré que j'ai pu me concentrer sur le moyen de le peindre sur un objet.

La seconde était un condensé de nombreuses manipulations dont le but était de créer, à partir d'une zone peinte sur la géométrie du sol, un attribut de température. Ce dernier indiquait précisément sur chaque point de l'objet quelle était sa valeur de chaleur. Ce qui me permis de générer un volume dont le rôle était de transmettre les informations de température depuis l'espace Géométrie jusqu'à celui des Dynamics qui gérait alors la création de la fumée. Les volumes sont des géométries 3D constituées de voxels, chaque voxel peut s'apparenter à un pixel à 3 dimensions. Au sein d'Houdini ils sont couramment utilisés pour transférer des attributs d'une géométrie à un solver de fluide, car ils ont l'avantage d'indiquer précisément dans l'espace les valeurs de tel ou tel attribut.

Le solver est quant à lui, le node qui permet de calculer le comportement du fluide. Ses différents paramètres d'entrée lui fournissent des informations issues d'un ou plusieurs objets, ainsi que de son environnement, tel que : la densité, la gravité, la vitesse, etc.

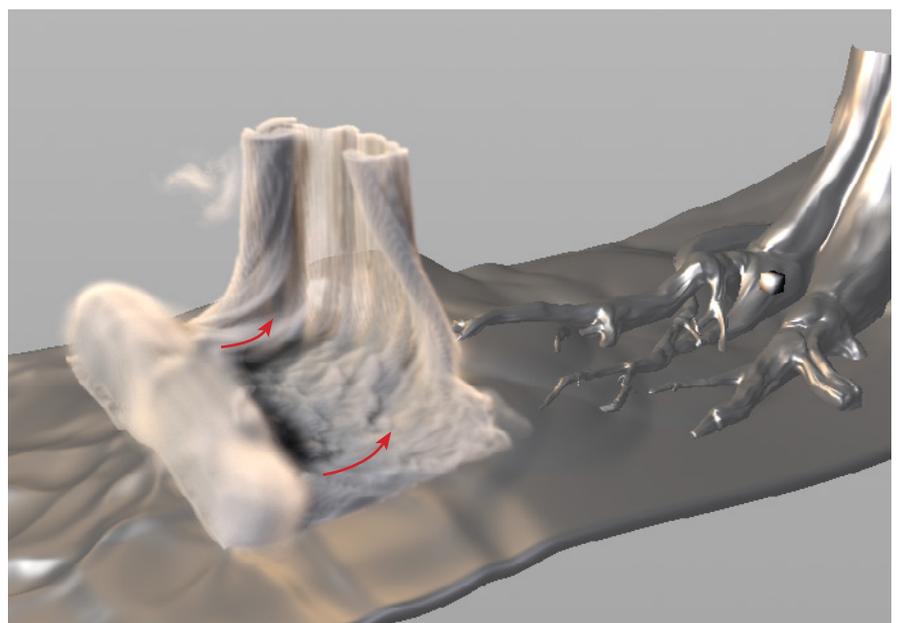
On peut ajouter autant de volumes que l'on désire à un solver, ainsi une fois que la fumée s'élevait j'avais la possibilité de créer un nouveau volume qui dirigeait le fluide dans l'espace 3D, en transmettant non pas des informations de température, mais de vitesse.



Température peinte sur l'objet.



Création du volume d'après la température.



En se référant aux images ci-contre, on peut constater que la fumée s'élève dans les zones chaudes. L'effet a été accentué pour l'illustration.

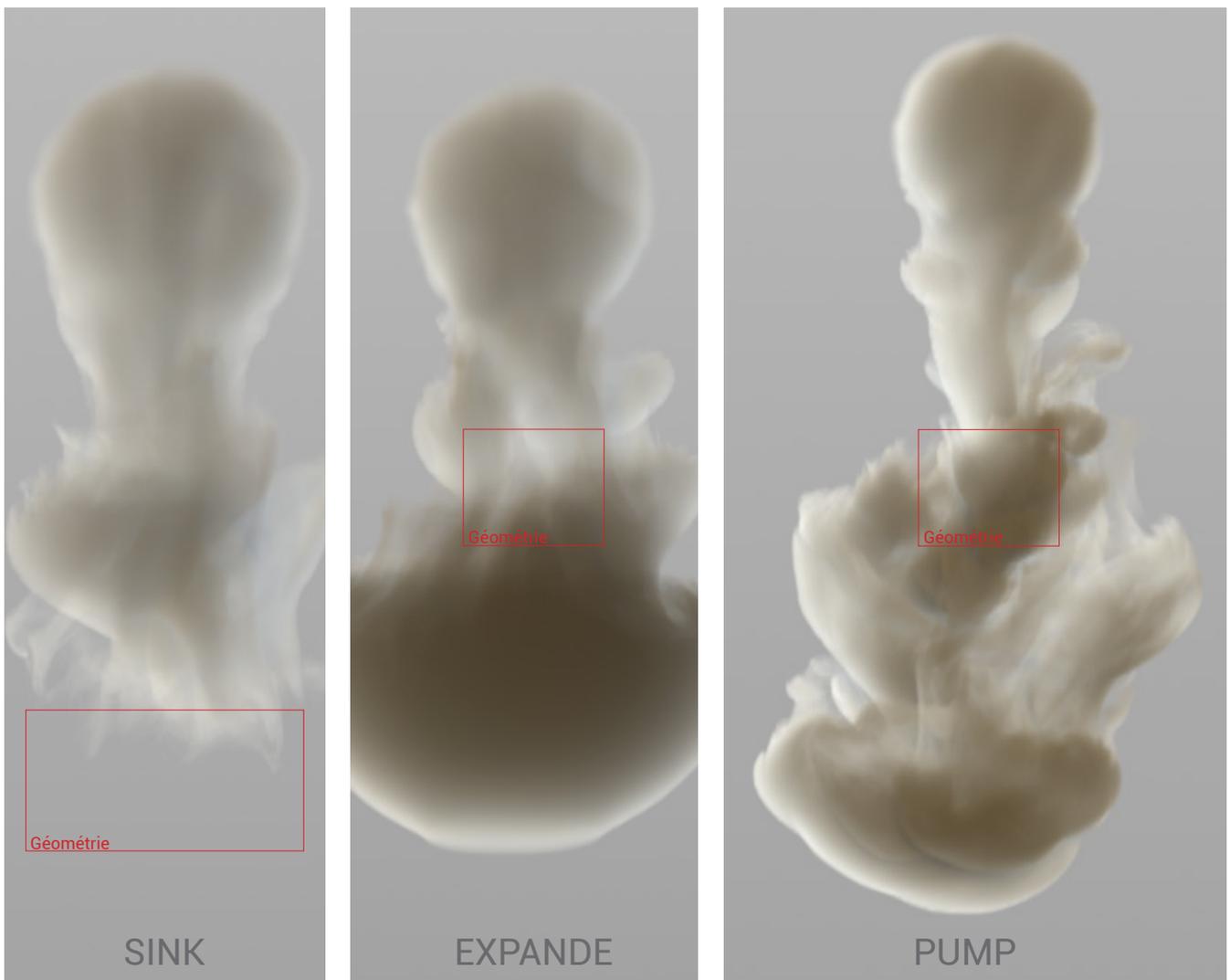
1.1.2 Avec des géométries 3D :

1.1.2.1 L' Interaction :

La fumée est constituée de particules solides en suspension dans les airs. Or tout solide peut entrer en collision avec un autre et ainsi perturber son comportement. C'est pourquoi, dans la réalité une des manières de modifier la trajectoire d'un fluide serait d'interagir physiquement avec, j'ai donc cherché à faire de même en 3D. L'onglet **Populate Container** propose différents outils (shelf tool) qui permettent à des géométries de donner divers comportements à un fluide, tels qu'en l'absorbant (Sink), en le répandant de chaque côté de la géométrie (Expande), en lui appliquant une vitesse secondaire (Pump), ou en créant une collision avec.



Ces icônes appelées Shelves Tools sont des raccourcis qui effectuent des connexions que l'on pourrait mettre en place de manière manuelle.



Houdini, en nous donnant accès au niveau le plus bas du logiciel, offre à l'utilisateur un contrôle total sur l'ensemble de ses outils.

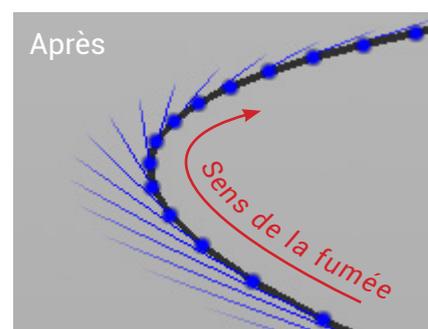
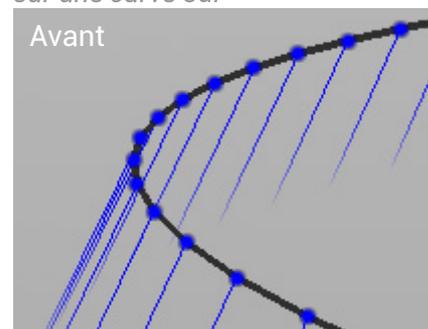
Ainsi j'ai pu déduire le fonctionnement du Shelf Tool **Sink From Volume** en observant les connexions nodales qu'il utilisait pour fonctionner. Ce qui me permit de constater une récurrence avec la méthode que j'avais employée pour transmettre mes informations de température à l'espace de Dynamics, et donc de comprendre l'aspect essentiel de l'utilisation des volumes.

1.1.2.2 Les Normales :

Jusqu'à présent, les méthodes utilisées ne me donnaient pas la possibilité de modifier l'orientation d'une fumée sur l'ensemble des axes 3D. Pour y remédier, la première solution qui s'imposa fut d'utiliser la forme d'une géométrie 3d comme guide.

J'avais précédemment expérimenté un système qui me permettait d'orienter les normales d'un objet dans le sens de ses courbes. Chacune d'elle pointait alors dans une direction précise, et pouvait faire office de vecteur. Il m'était alors possible d'utiliser ces vecteurs pour générer un volume qui guiderait la fumée le long des normales.

Exemple de redirection de normale sur une curve 3d.



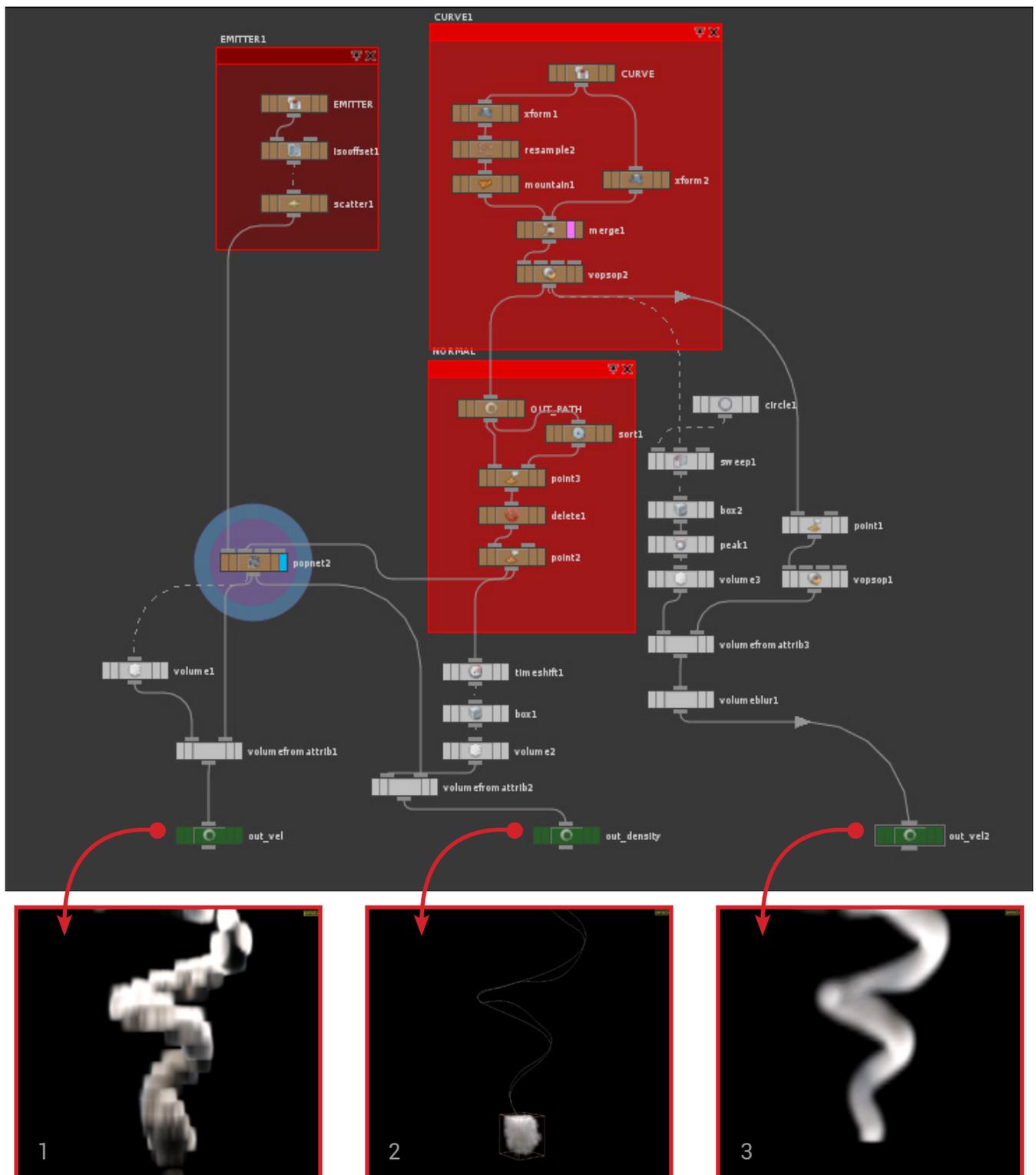
Une astuce consistait à multiplier la quantité d'information de normales en créant une géométrie autour des curves, puis en répandant un nuage de point à l'intérieur, je transférais ensuite sur chacun de ces points les informations de normale. Ainsi l'ensemble d'entre eux indiquait le sens que la fumée devait suivre.

(ci contre : une fumée dirigée par des curves).

1.1.3 Avec des particules :

1.1.3.1 Les Volumes :

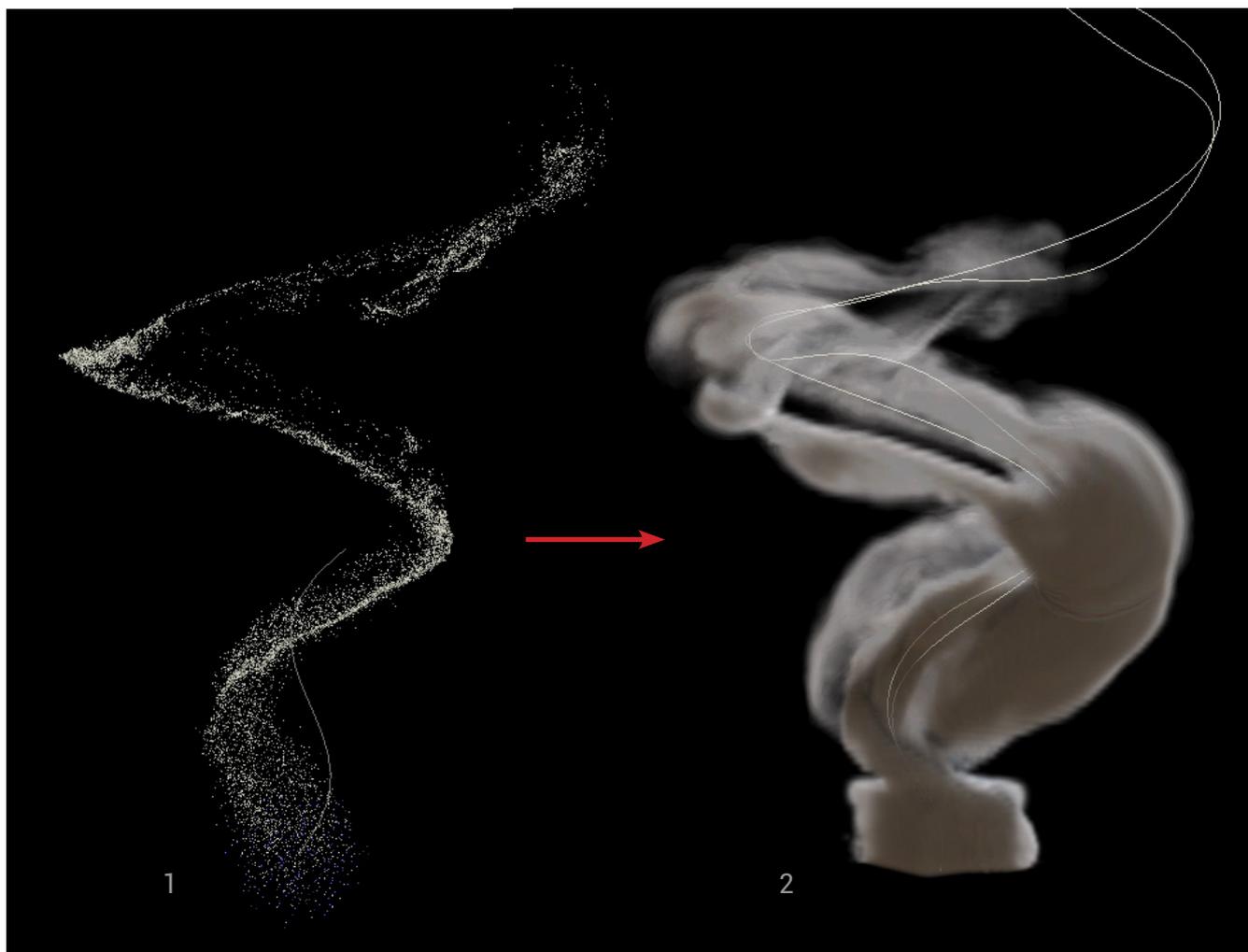
Une autre méthode, à peu près similaire dans sa mise en place, a été de diriger le fluide grâce à un champ de particules dynamiques qui avançait le long d'une curve. Afin que l'effet fonctionne correctement, j'ai utilisé 3 volumes, chacun étant exporté vers le solver de fumée pour servir de guide.



Le premier capturait le mouvement des particules dans un volume de vélocité afin d'orienter le fluide.

Le second convertissait l'émetteur de particules en volume de densité, alors chargé d'émettre de la fumée.

Et enfin le troisième était un volume de vélocité annexe, qui transformait en volume la curve servant aux particules de guide, et ce, dans le but d'indiquer à la fumée de monter.



Ci-dessus : le champ de particules ayant servi à générer le volume de vélocité¹, puis la simulation de fumée².

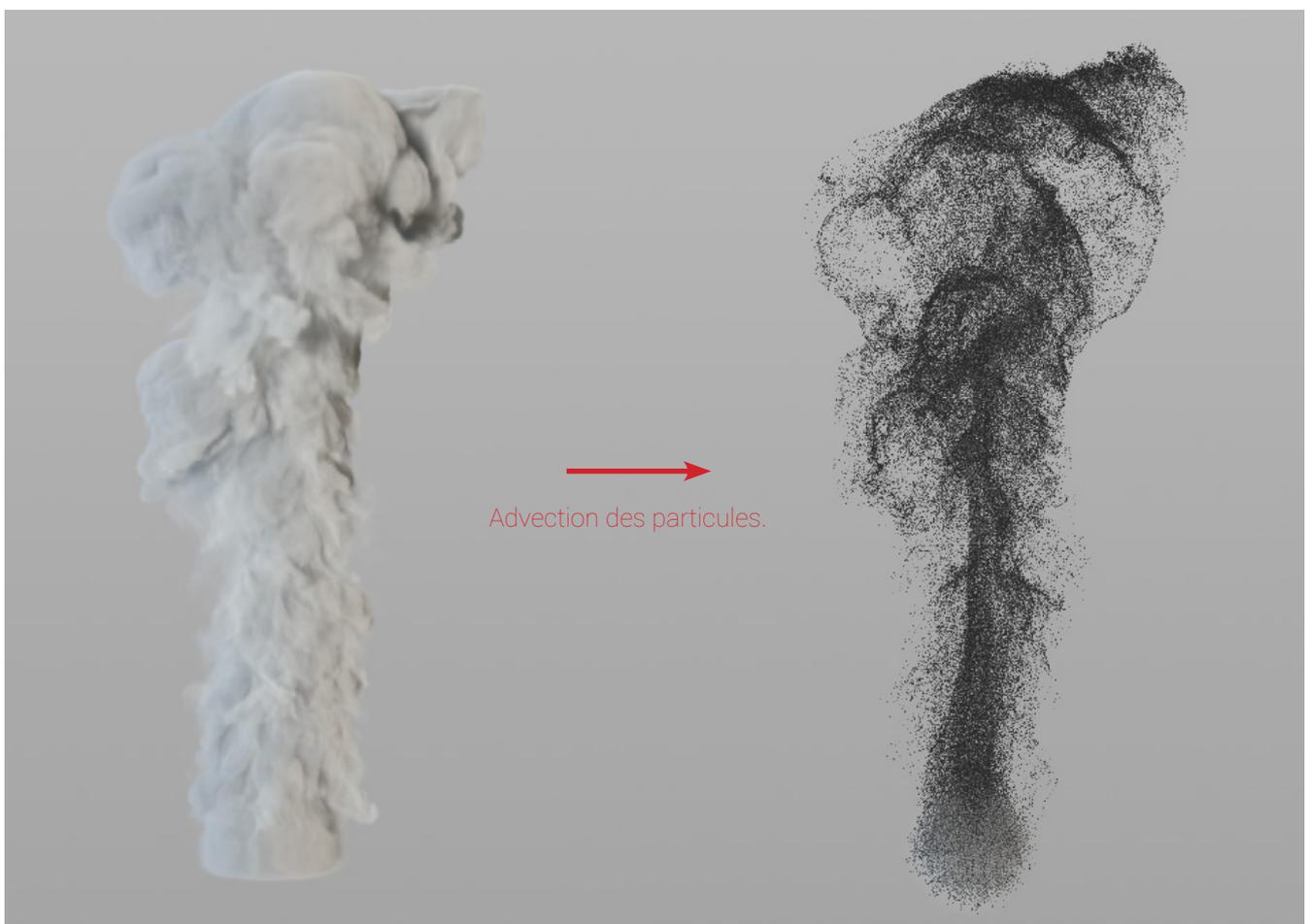
Pour conclure, cette méthode semble la plus efficace que j'ai utilisée, car le fluide est en mesure de suivre de manière assez fine la trajectoire qu'on lui donne tout en gardant une attitude naturelle. De plus les divers paramètres du solver de fumée permettent une maîtrise poussée du comportement donné au fluide. En combinant cette technique et les précédentes, je pense donc être capable d'obtenir un niveau de contrôle correct pour les simulations à venir.

1.1.3.2 L' Advection :

De la même manière que dans le logiciel Maya, il est possible dans Houdini de faire suivre le mouvement d'une fumée à des particules, on appelle cette opération : advecter des particules. Au départ j'utilisais l'advection dans le but d'augmenter la résolution d'une fumée : les particules allaient alors récupérer dans leurs déplacements des informations très précises sur l'ensemble des micros-mouvements qui s'effectuaient à l'intérieur du fluide. Ces informations étaient ensuite réutilisées dans un solver pour éviter toute perte de détails lorsque la fumée se dissipait (Pour plus de précision sur les méthodes d'advection, se référer à la publication de Magnus Wrenninge, voir le lien, page 41).

J'ai détourné ce procédé afin d'être en mesure de corriger la dynamique d'une simulation de fluide, en apportant des modifications dans celle du champ de particules.

Il me suffisait alors de créer un cache des particules, puis de changer leurs trajectoires, avant de les réutiliser pour générer une nouvelle fumée avec un aspect légèrement différent.



1.2 RENDRE UNE FUMÉE :

Sur Houdini on peut rendre de nombreuses passes pour le compositing d'une fumée. Le shader Pyro, utilisé pour la création d'explosions et de flammes, est construit de sorte à pouvoir isoler la fumée des flammes et générer un masque pour ces deux couches (*ci-contre*).

De surcroît, ce même shader contient une option qui permet d'obtenir un rendu similaire à ce qu'affiche le logiciel en temps réel. Ce qui est une véritable respiration après avoir été de nombreuses fois frustré par le rendu des fumées dans Maya qui ne correspondaient pas à celui du viewport. Le

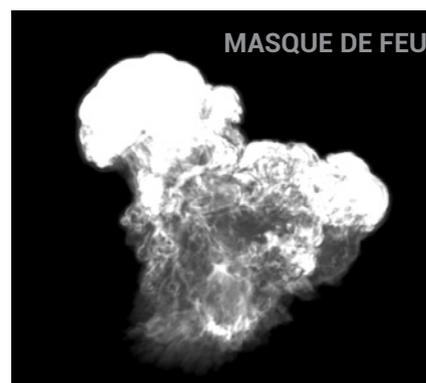


fait d'avoir un aperçu en

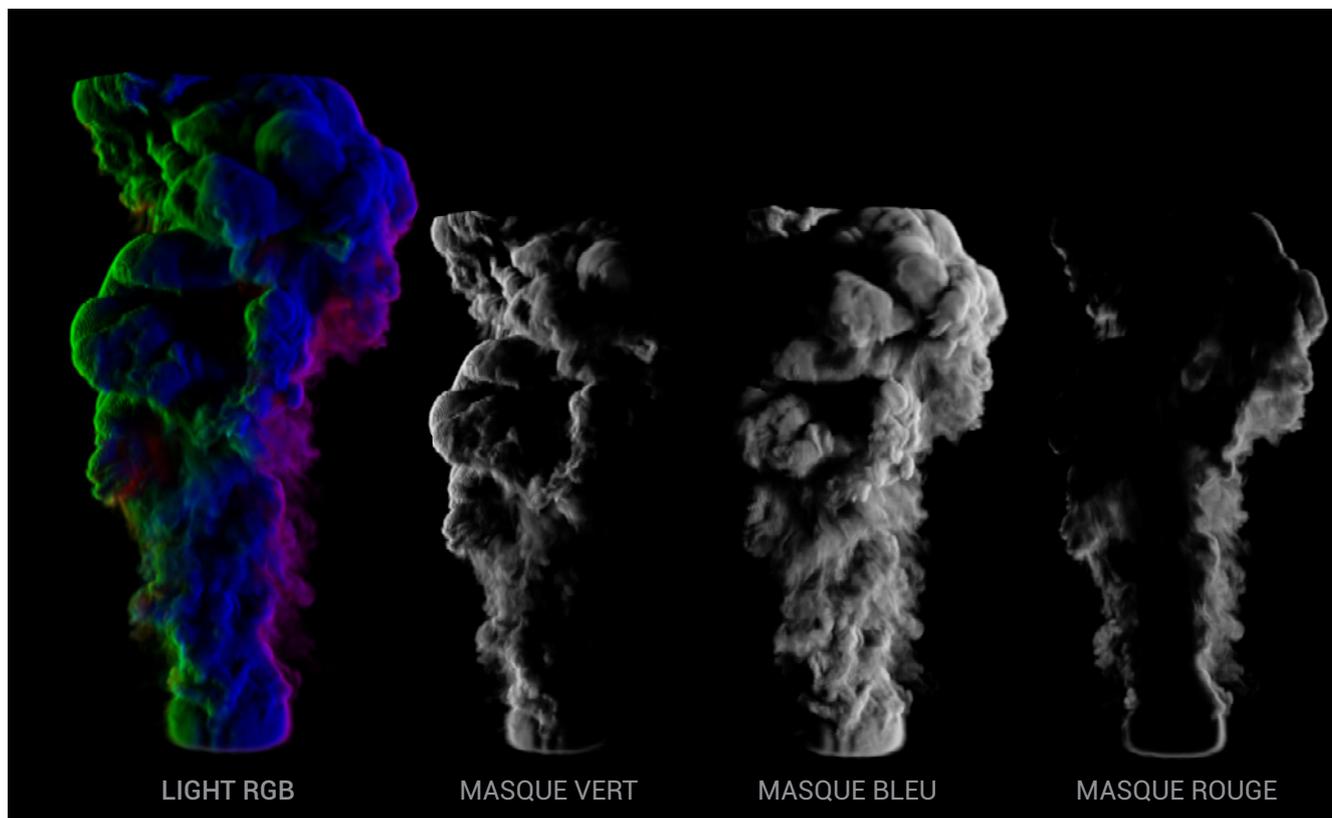
temps réel de sa fumée permet un contrôle artistique assez poussé, de la même manière qu'avec le plugin Fume FX disponible sur Maya et 3DS Max.

La passe de profondeur quant à elle est difficile à obtenir, car les voxels ne sont pas correctement évalués par les méthodes traditionnellement utilisées pour la créer. Une solution existe, qui consiste à produire un fichier de profondeur capable d'être interprété par le module de compositing d'Houdini. Mais je n'ai pas eu le temps de me pencher davantage sur la question pour le moment.

La version 12.5 du logiciel intègre un nouveau type de lumière qui permet d'éclairer à partir d'un volume. Ainsi j'ai été capable d'illuminer une fumée par son feu d'origine, en générant à partir des zones de combustion un volume qui me servait de source d'éclairage.



La passe indispensable lorsque l'on rend une fumée est celle de Light RGB. Elle consiste à placer 3 sources de lumière autour du fluide (une rouge, une verte et une bleue). Chacune d'elle servira alors de masque au compositing, nous offrant ainsi la possibilité de retoucher l'intensité lumineuse dans ces zones séparément. Idéalement on crée un éclairage composé d'une feel light, d'une back light et d'une key light, afin de faire ressortir le volume du fluide.



2 : LES PARTICULES

Le court métrage comptait deux personnages, l'un était un papillon et l'autre un champ de particules, qui sur la quasi-totalité de la durée du film déambulait dans la végétation.

Ces particules devaient alors suivre une trajectoire précise, mais j'ai très vite fait le choix de leur laisser une certaine liberté de mouvement, car dans le contexte de cette nature qui s'éveille, les particules devaient posséder un semblant de vie. Or un mouvement trop précis, trop mécanique serait allé à l'encontre de cette vision.

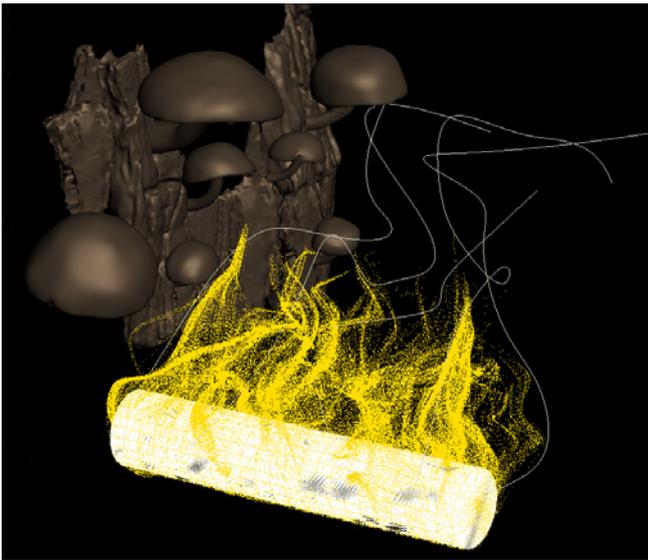
2.1 DIRIGER LES PARTICULES :

J'ai naturellement cherché à adapter un système que j'avais déjà exploité sur les fumées pour diriger les particules. Il consistait à indiquer un itinéraire à une simulation par le biais des normales d'une curve. Même si la logique était la même, la mise en place, elle, était totalement différente.

Pour contrôler des particules en manipulant leurs attributs, l'outil le plus efficace est le node VOPPOP. Il met à disposition des noeuds de type VEX.

Le VEX est un langage d'Houdini qui a la capacité d'être très rapide à calculer pour la machine. C'est pourquoi ces nodes peuvent effectuer des opérations sur des centaines de milliers de particules de manière optimisée.

La plupart des nodes VEX que j'ai utilisés ont des fonctions très simples, tel que : multiplier, soustraire ou diviser des valeurs, alors que d'autres, plus complexes, mesurent des distances, effectuent des produits en croix ou créent des boucles, etc.



Ci dessus : le champs de particules se déplaçant le long des curves.

Pour transmettre les informations de normales aux particules lorsqu'elles étaient à proximité de la curve, j'ai utilisé le node Point Cloud Open.

Ce node de type VEX compare la position d'un nuage de point (dans notre cas les points de la curve) à celle d'une géométrie (les particules) dans le but de transférer des attributs de l'un vers l'autre.

Afin de confiner un attribut spécifique dans un nuage de point, et ainsi être

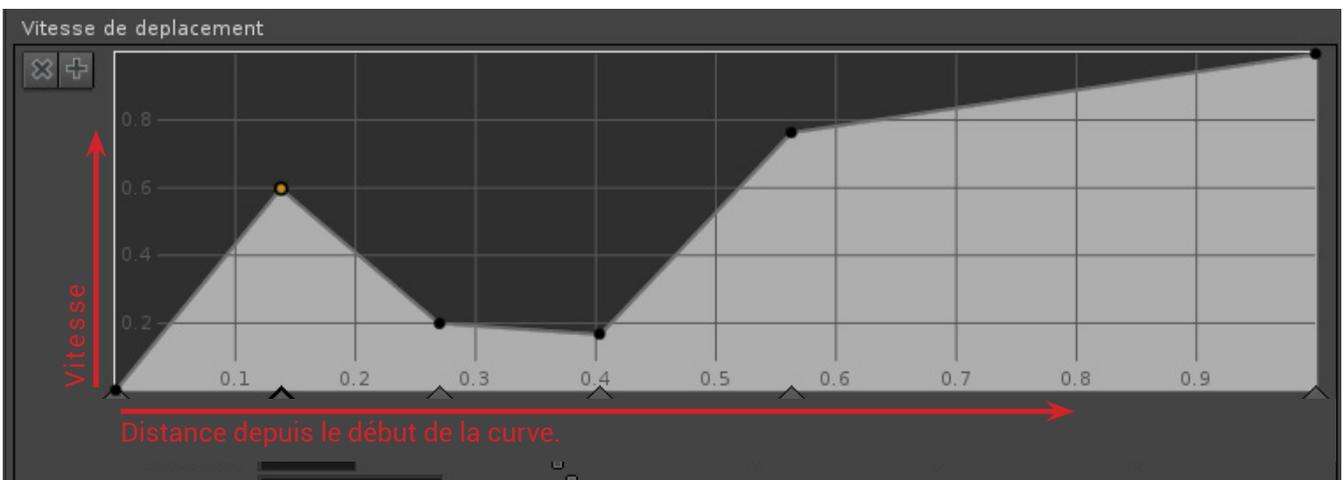
capable de le manipuler, on utilise à la suite du Point Cloud Open des nodes Point Cloud Filter.

Ainsi, j'ai isolé trois attributs pour créer mon effet :

- L'un déduisait la distance entre chaque point de la curve et la longueur totale de cette dernière, ce qui permettait à terme de contrôler la vitesse des particules suivant leurs positions sur la curve.

- L'autre indiquait l'emplacement des points sur la curve et était employé pour diverses opérations mathématiques.
- Et enfin le dernier représentait les normales et communiquait aux particules la direction à suivre.

Une fois que les particules avaient un comportement correct, j'ai ajouté quelques connexions au node VOPPOP qui me permettaient de créer une interface que j'allais pouvoir utiliser pour régler leurs comportements au cours de leurs déplacements (vitesse, fidélité à la curve et distance avec la curve). Il se présentait sous la forme d'une courbe (ramp) dont l'abscisse servait à régler la vitesse et l'ordonnée la distance depuis le début de la curve.



L'annexe : Diriger les particules (p.57), revient sur les détails techniques de la méthode employée pour diriger des particules avec le Point Cloud.

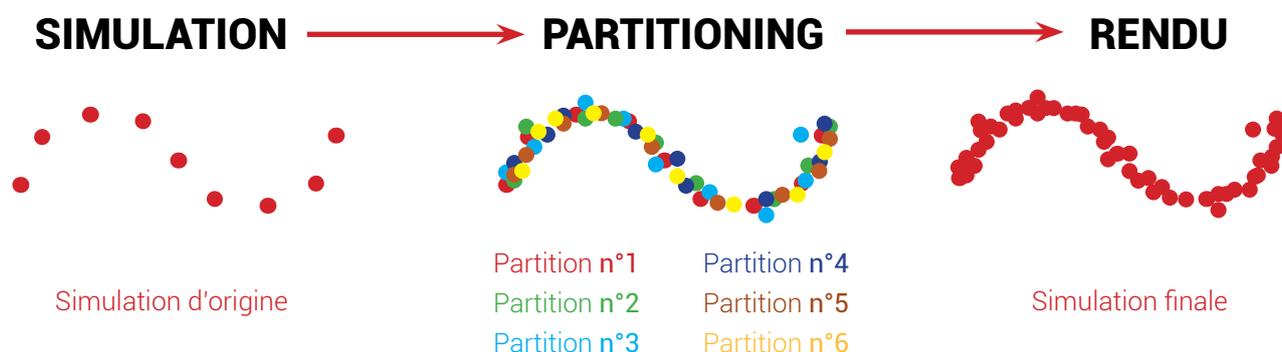
2.2 : RENDRE LES PARTICULES :

2.2.1 Le Partitioning :

Pour le projet de film de trois semaines, j'étais tenu de fournir 6 plans de particules, dans chacun d'eux la simulation devait paraître différente, et pour des raisons d'esthétisme, ces champs de particules nécessitaient des millions d'entités. Il me fallait aussi mettre en place deux plans de destruction, ainsi qu'une fumée pour la première séquence. Les outils permettant de

créer ces effets étaient prêts avant le début de la production, il ne manquait alors plus qu'à les adapter pour chaque plan. Étant donné la quantité de simulations à lancer, et de passes à rendre, je devais me constituer un workflow optimisé.

Afin de créer une animation comportant des millions de particules, Houdini possède un système qui permet de dupliquer une simulation autant de fois que demandé, dans le but de les superposer les unes sur les autres. Sur chacune d'elles, il modifie légèrement la position des particules. On appelle cette technique le partitioning.



L'avantage est qu'Houdini n'a pas à calculer une nouvelle simulation pour chaque partition, ce qui fait gagner un temps non négligeable.

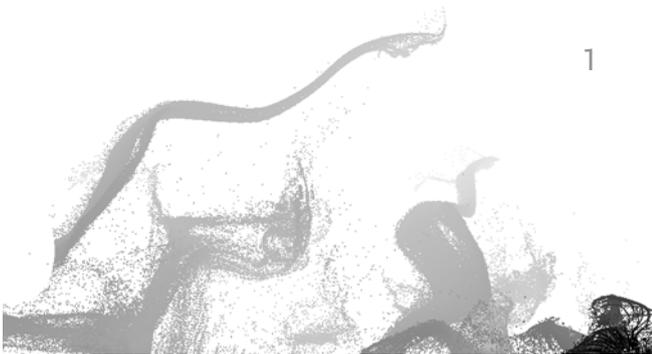
On retrouve un procédé similaire dans les logiciels 3Ds Max et Maya, avec le plugin Krakatoa (à l'époque il n'était pas encore sorti sur Maya).

Afin de demander encore moins de calcul à la machine, j'avais aussi mis au point un outil qui supprimait les particules ne figurant pas dans le champ de la caméra.

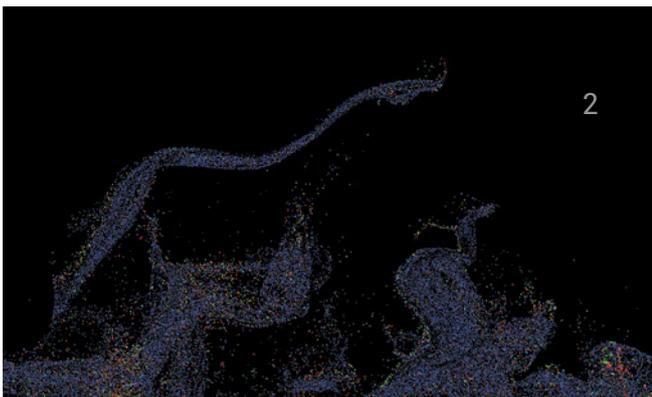
2.2.2 Les passes :

Durant cette année mon but était avant tout d'apprendre Houdini, mais j'étais curieux de recourir à d'autres logiciels pour comparer et diversifier ma palette d'outils. J'ai donc très vite décidé d'exporter mes particules dans 3Ds Max afin d'effectuer le partitioning sur Krakatoa. Ce plugin me faisait rêver depuis des années, car ses rendus sont sublimes, et 3Ds max à l'avantage de posséder une passerelle directe entre lui et Maya. Or ce dernier avait été

employé pour réaliser les décors et le Lighting du projet. Grâce à ce système, je pouvais donc importer facilement les décors et y intégrer les particules. Lorsque je devais envoyer des éléments depuis Maya vers Houdini, j'utilisais le format Alembic. Développé par ILM et Sony Pictures Imageworks, il permet d'exporter des scènes complexes d'un programme à l'autre en faisant un bake des animations de chaque géométrie. Mais il m'avait à l'époque causé de nombreux problèmes dans Houdini, si bien qu'à présent je me sers davantage du format fbx. Pour exporter les particules depuis Houdini vers 3Ds Max



1



2



Dans une logique d'efficacité, Krakatoa offre la possibilité de créer des passes en un clic. On peut donc générer une Zdepth¹, une Motion Vector², et tout autre passe sans avoir à effectuer des opérations complexes.

le problème était différent, il ne semblait pas exister de format natif qui soit compatible entre l'un et l'autre. J'ai finalement utilisé un plugin créé par les développeurs du logiciel Realflow, qui une fois installé sur Houdini et 3Ds Max, pouvait exporter et importer les particules.

La raison de l'existence de ce plugin est que Realflow est avant tout un logiciel de simulation de fluides et qu'il ne disposait pas au départ de moteur de rendu, or ses fluides sont générés à partir de particules, et dans certaines situations, on a besoin de les rendre, pour créer de l'embrun ou des gouttelettes.

C'est pourquoi lorsque l'on utilise Realflow, on effectue de nombreux rendus avec Krakatoa (même si de plus en plus sont effectués sur le moteur Maxwell), il faut donc exporter les particules d'un logiciel à l'autre. Krakatoa a l'avantage d'être facile à prendre en main et très rapide. Ce moteur donne la possibilité d'appliquer des shaders complexes à un

champ de particules, tel que la translucence. On peut y régler de manière intuitive l'épaisseur des particules, leurs transparences ou la manière dont les ombres et les lumières se diffusent au sein d'elles. De plus, il dispose d'une interface qui propose une visualisation en temps réel des modifications apportées aux shaders.



Çi-dessus un teste de rendu de millions de particules advectées dans une fumée FumeFX de 3Ds Max.

Il est aussi parfaitement intégré à 3Ds Max, qui permet d'utiliser son système Pflow afin de modifier les shaders des particules suivant certaines conditions telles que : l'âge, la vitesse, la position, etc..

PFlow est une interface nodale qui pourrait s'apparenter à celui d'Houdini, je me suis donc rapidement familiarisé avec.



Çi-dessus le plan 4 du film. Texturing & modeling : Eva Virlovet et Sophie Garrigues, Lighting : Eva Virlovet. J'y ai effectué le compositing et les VFX.

Bien que Krakatoa ne soit pas Multithreadé* à la différence de Houdini (* il n'utilise qu'un seul coeur sur la totalité que le processeur contient), il est très optimisé et permet de faire le partitioning et le rendu en un temps record. J'avais alors tout intérêt à m'en servir, car il restait malgré tout le plus efficace. Dans la mesure où rendre et partitionner les particules dans 3Ds Max au lieu de le faire dans Houdini me demandait une manipulation supplémentaire, et que de nombreux plans contenaient des particules, j'avais décidé de ne pas créer une simulation différente pour chaque séquence. C'est pourquoi j'ai mis en place un émetteur de particule qui était branché à suffisamment de courbes pour créer des motifs d'émission différents. Une fois la simulation mise en cache, il ne me restait plus qu'à en extraire certaines parties afin de les disposer dans le plan suivant les besoins. Le contrôle n'était pas aussi précis que si j'avais créé une courbe distincte pour chaque plan, mais j'ai pu gagner une semaine de travail.

Pour conclure, malgré le temps que j'ai mis à créer cet effet, il m'a donné la possibilité de me former au VEX BUILDER, à Fume FX et à Krakatoa, et ainsi, de découvrir à l'intérieur et au-delà d'Houdini de méthodes pour contrôler un champ de particules. Utiliser des normales est un bon moyen pour diriger précisément une simulation, d'autant plus que le degré de liberté accordé aux particules permettait de leur donner un semblant de vie, car le mouvement semblait organique et non mécanique. Ainsi l'outil a pu de lui-même me proposer des motifs que je n'aurais alors pu imaginer moi-même.



Une de mes principales références dans ce travail était les effets de particules Houdini du film *Les cinq légendes* de Dreamwork Animation.

Dans les mois précédents le début du projet nous avons décidé de ne pas ouvrir le cocon de manière naturelle. Il était alors constitué d'une matière calcaire et devait se fissurer afin de laisser sortir le papillon. Or l'équipe en charge de modéliser et texturer le cocon avait conclu, au moment de la production, de lui donner un aspect réaliste et organique, ce qui remettait en question cet effet.

J'avais mené mes recherches de sorte à créer un outil capable de mettre en place des fractures dans des zones précises de la géométrie, ainsi que de contrôler leurs tailles et leurs nombres. Il permettait de sélectionner très précisément quelle pièce fracturée tomberait et à quel instant, et offrait donc une supervision totale de la destruction.

Parallèlement, j'ai effectué des recherches sur un effet annexe, qui avait pour objectif d'apporter du détail au plan. Il s'agissait de créer une membrane à l'intérieur du cocon qui devait se déchirer lors de l'éclosion.

3.1 BRISER LE COCON :

3.1.1 Préfracturer :

Chaque matériau lorsqu'il est détruit réagit de manière différente, ainsi du bois se brisera en lamelles, alors que du béton se fractura en bloc. Préfracturer un objet permet de contrôler la forme qu'auront ses débris, cette étape est donc primordiale.

L'outil qu'utilise Houdini pour engendrer ses fractures se base sur le diagramme de Voronoï. Qui, malgré sa rapidité à calculer, ne produit pas une très grande variété de formes si on ne l'utilise pas correctement.

Le diagramme de Voronoï va créer à partir de points répartis dans l'espace un ensemble de cellules. Il trouve différentes applications, allant du calcul de trajectoire à la création d'effets de mosaïques pour l'infographie.

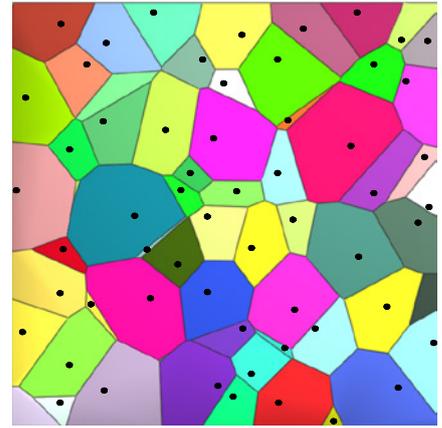
Dans Houdini chaque cellule représente alors une des faces d'un des débris

constituant l'objet fracturé. Afin de les générer, il lui faut une géométrie et un nuage de points.

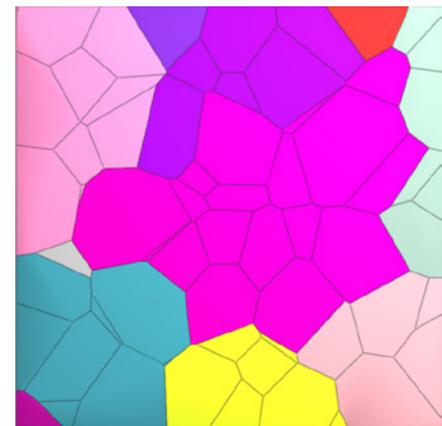
On a la possibilité d'obtenir ce dernier de différentes façons, la plus simple étant de convertir le mesh à fracturer en volume, puis de répartir à l'intérieur un ensemble de points de manière automatique avec le node Scatter.

Le node VornoiFracture alors relié à la géométrie initiale et au nuage de point se chargera de générer les fractures.

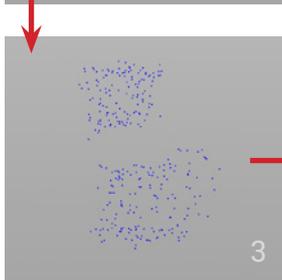
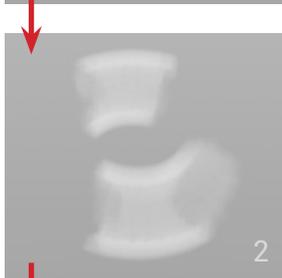
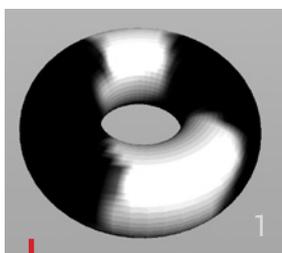
Les formes que crée le Vornoi sont très reconnaissables et régulières (semblables aux motifs d'une girafe), pour plus de réalisme il faut donc produire des variations dans leurs apparences. Grâce au paramètre Cluster, on peut regrouper plusieurs cellules ensemble afin d'en créer une nouvelle. La forme obtenue est ainsi plus complexe.



Cellules de Voronoï, avec les points qui ont servi à les créer.



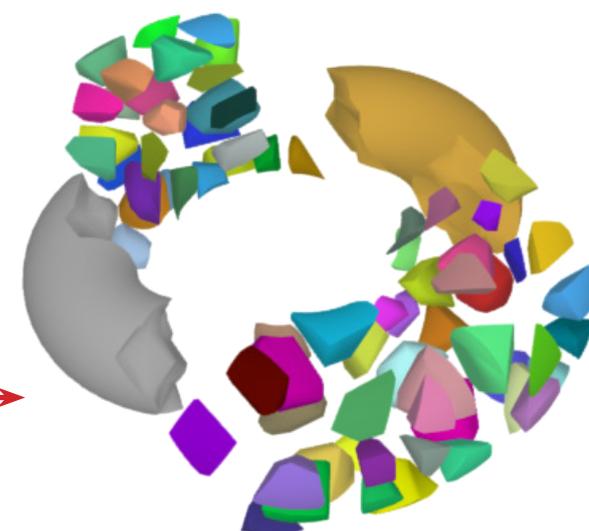
Exemple de cellules de Voronoï réunis par un Cluster.



Pour mettre en place plus de fractures, il suffit de répandre plus de points. Afin de contrôler l'endroit où ils seront répartis, j'ai expérimenté deux techniques :

La première consistait à peindre¹ sur l'objet une couleur qui servait à générer un volume², ce dernier indiquait alors au noeud Scatter³ où reprendre les points.

Peindre directement sur un mesh peut causer des problèmes si sa topologie est modifiée au cours de la production, car le node paint se sert des points de la géométrie comme référence pour

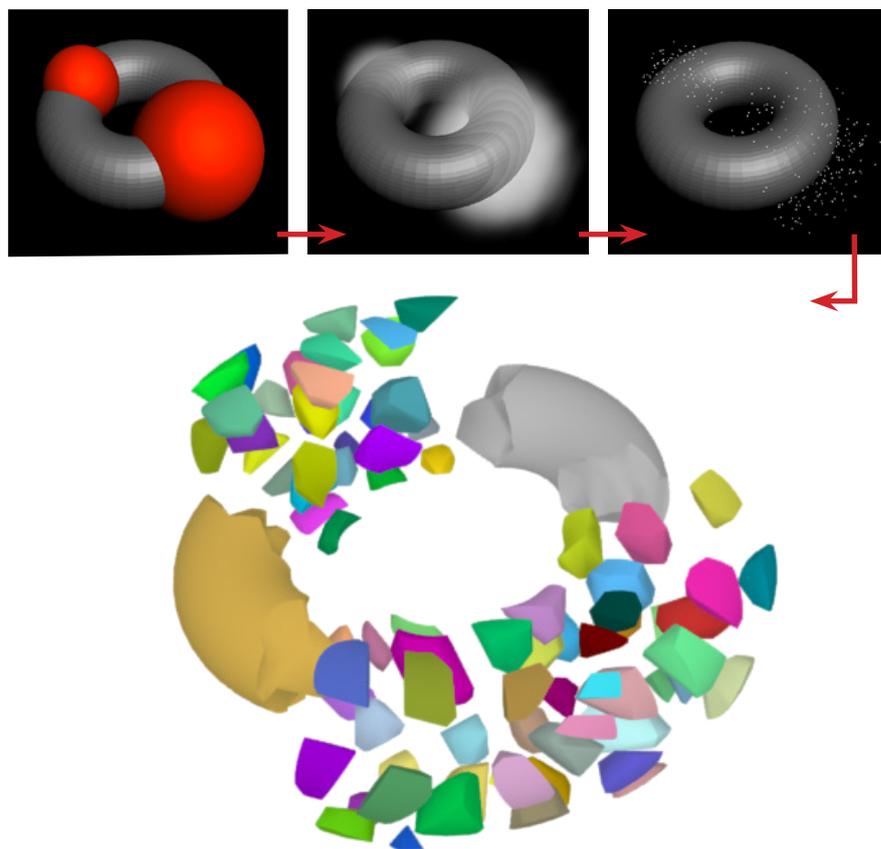


appliquer la couleur. Or, chaque point dispose d'un numéro d'identification qui permet au node de savoir précisément à quel point appliquer telle ou telle valeur chromatique. Et tout changement de typologie implique de modifier la position de ces numéros et donc de la couleur.

Employer un node paint rendait l'outil moins procédural et par conséquent moins intéressant à utiliser pour le projet de court métrage. Car ayant entrepris mes recherches de fracture avec une géométrie de cocon temporaire, j'allais être amené à appliquer cet effet à une autre dans les mois à venir.

C'est pourquoi, dans l'optique d'apporter le moins de rectifications possibles lors d'un changement de géométrie, j'avais expérimenté une nouvelle manière de distribuer les points

de fracture : cette méthode reposait sur le même principe que la précédente, à la différence que le node paint qui avait pour fonction de répartir les points était remplacé par une géométrie primitive. Ainsi si l'objet à fracturer était amené à changer, le nuage de point quant à lui restait à la même position. Ce qui me permettait de travailler sur une représentation simplifiée du mesh, et de la remplacer ultérieurement par la version définitive.



3.1.1.1 Contrôler la taille des débris :

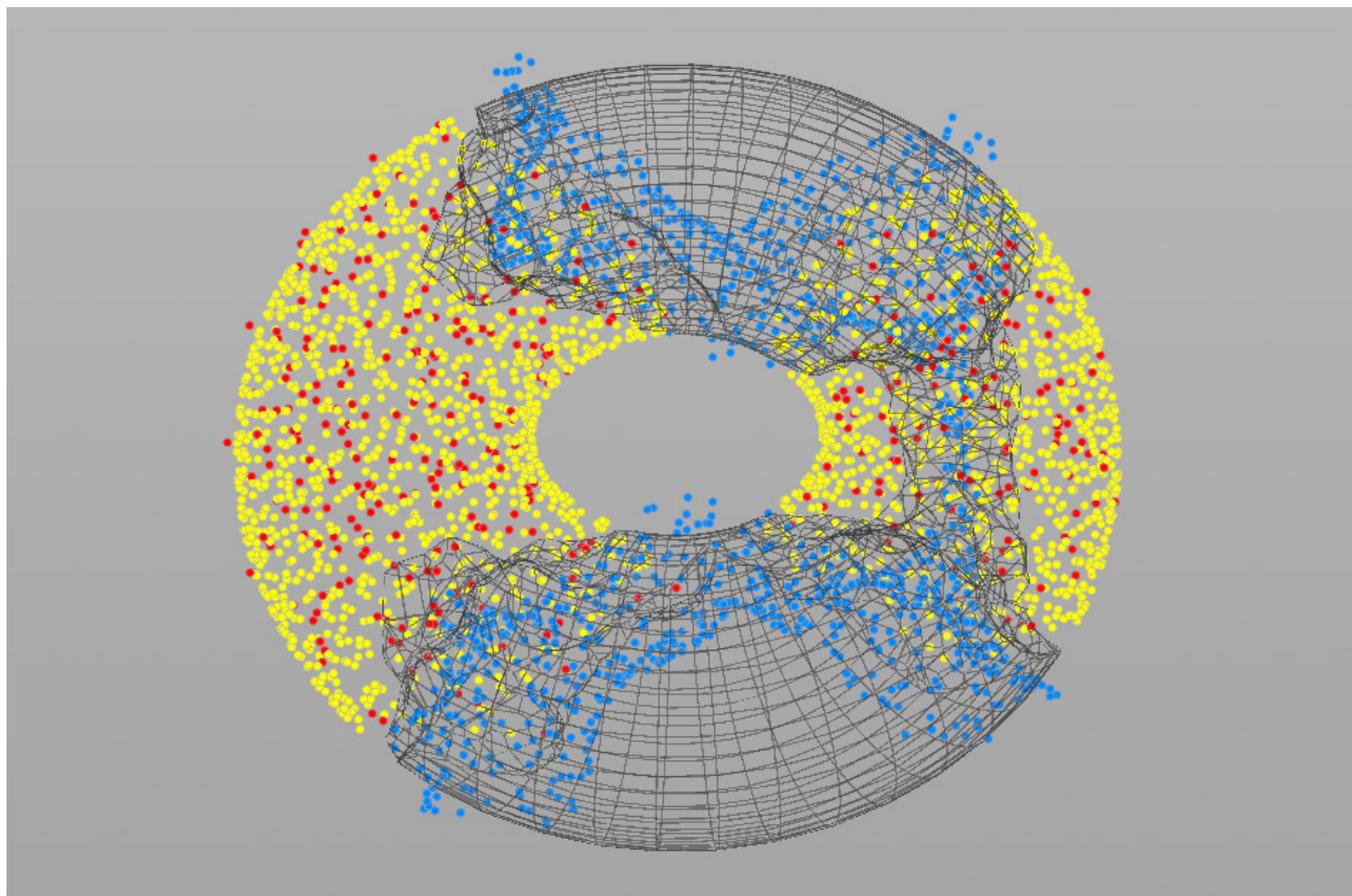
Afin de donner une échelle différente aux débris issus de l'intérieur ou de la surface de l'objet fracturé, on peut utiliser le node Voronoi Fracture Point. Il répartit les points du Voronoï en trois catégories :

Points de surface : placés à la surface du mesh.

Points d'intérieur : situés dans la masse de l'objet.

Et points d'extérieur : disposés à la limite entre la partie fracturée et la partie intacte.

Il permet de régler le nombre de points répartis dans chaque catégorie manuellement. Par conséquent, on peut contrôler le nombre de fractures qu'il y aura à la surface, à l'intérieur et à la limite entre l'objet et les débris.



Exemple de répartitions des points de fracture avec le node Voronoi Fracture Point

Si on veut donner l'impression qu'une géométrie s'effrite on placera moins de points à l'intérieur de l'objet, et plus à l'extérieur, ce qui engendra de grandes pièces dans sa masse et de multitudes de petites à sa surface.

3.1.1.2 Optimiser :

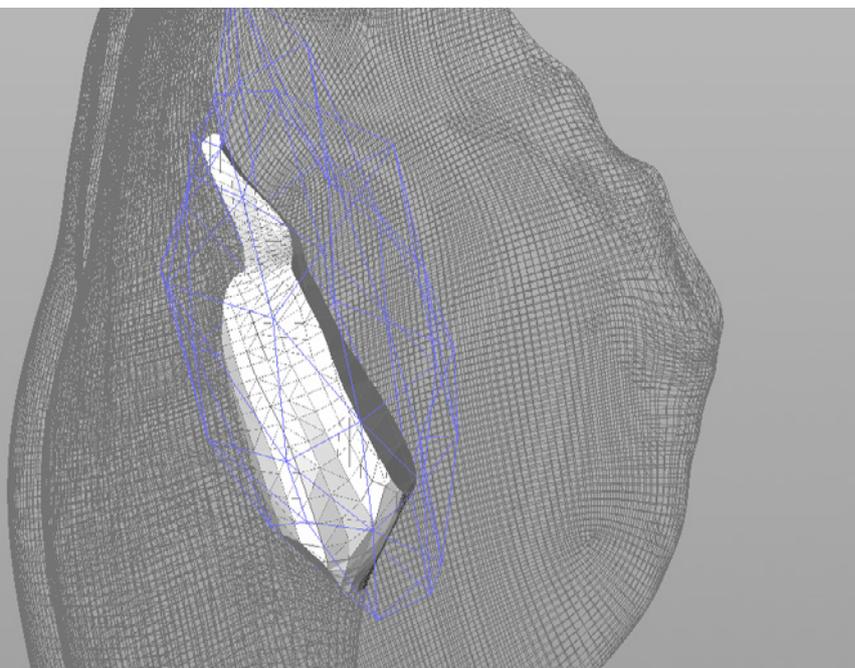
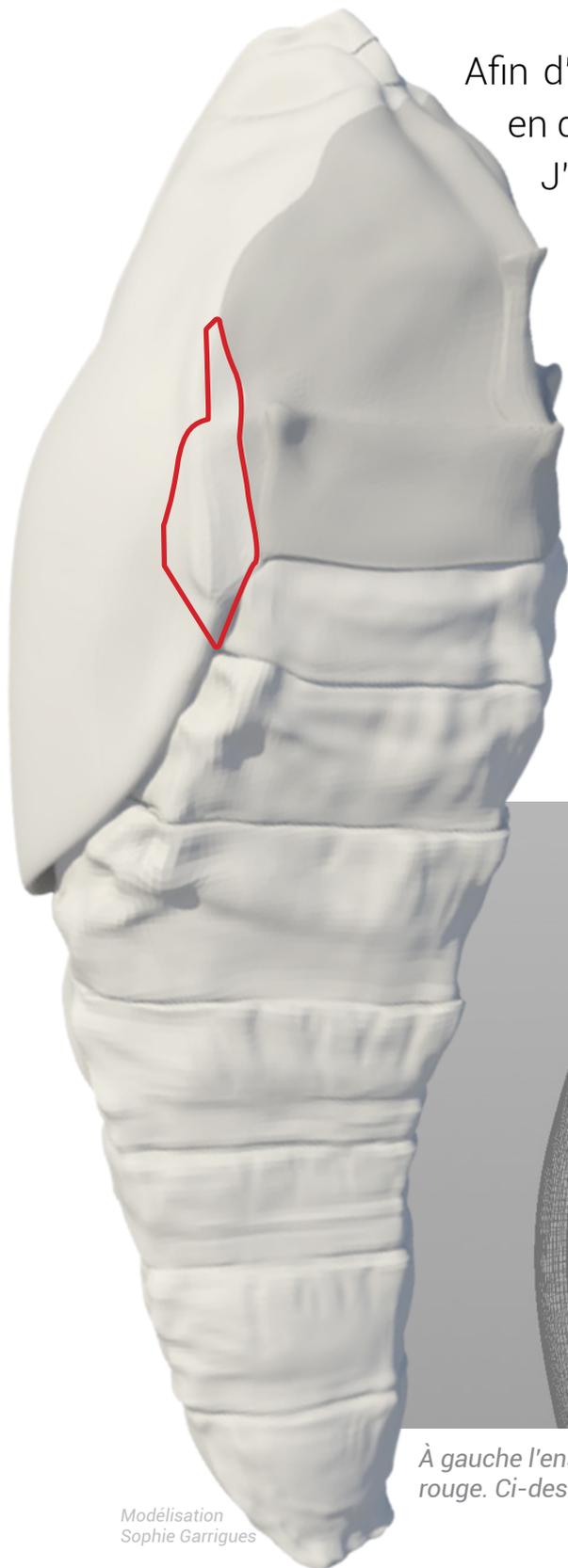
Lors de mes recherches en amont du projet j'avais tendance à pré-fracturer l'ensemble de l'objet, même si une seule partie devait être détruite. Je travaillais alors sur une géométrie de cocon temporaire qui comportait peu de polygones. Or le cocon utilisé pour le film en disposait d'un nombre beaucoup trop important, ce qui demandait à l'ordinateur des efforts considérables pour effectuer la moindre opération.

Le nombre de polygones ne posait pas de réels problèmes pour créer de grandes fractures, mais au fur et à mesure du temps le reste de l'équipe sou-

haitait des débris de plus en plus petits, ce qui impliquait plus de polygones à simuler et à contrôler. Et les débris générés n'étaient malheureusement pas assez petits pour être créés avec des particules, ce qui aurait permis d'optimiser la scène. Mon principal souci avec ce plan, était la taille des caches, qui devaient cohabiter avec celles des particules et du partitioning.

Afin d'optimiser ma scène j'ai donc séparé le mesh en deux parties, l'une étant fracturée et l'autre non.

J'avais utilisé à l'époque le node Break, qui divise une géométrie au niveau de l'intersection entre l'objet à découper et un autre. Ce qui me permettait de sélectionner précisément la zone où j'allais créer mes fractures. Mais ce système ne fonctionne correctement que lorsque les deux meshes contiennent une densité de polygone à peu près similaire. Ce qui une fois encore pose un problème d'optimisation, car le calcul est alors plus long. Le reste du temps des bugs se créent au niveau de la découpe.



À gauche l'ensemble de la Géométrie du cocon et la zone de simulation isolée en rouge. Ci-dessus la géométrie de simulation isolée par le mesh du node Break.

Modélisation
Sophie Garrigues

Pour remédier à ça, le node propose un système qui triangulise le mesh, le calcul semble alors plus optimisé pour le logiciel. Mais cette solution modifie la topologie, et les Uvs de l'objet ne correspondent donc plus.

Le fait d'utiliser une géométrie personnalisée pour créer la zone dans laquelle les débris se génèrent peut être intéressant à partir du moment où l'on souhaite avoir une forme très précise de l'endroit où se créeront les fractures, ce qui n'était pas le cas. Aujourd'hui j'ai recours à un autre système, plus procédural, et plus précis au niveau de la découpe qui consiste à fracturer le mesh en différentes pièces, puis à utiliser l'une des pièces engendrées comme champ de fracture.

3.1.2 Simuler :

Une fois le cocon fracturé, il fallait alors rendre dynamiques les débris en leur appliquant collision et gravité.

3.1.2.1 Les Solvers :

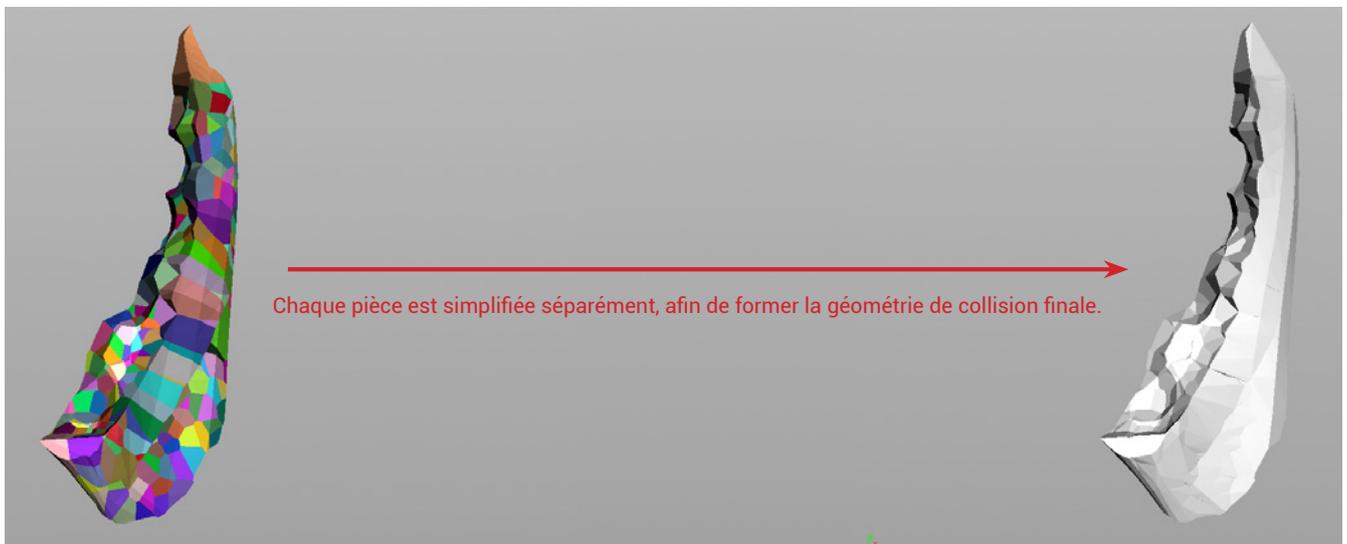
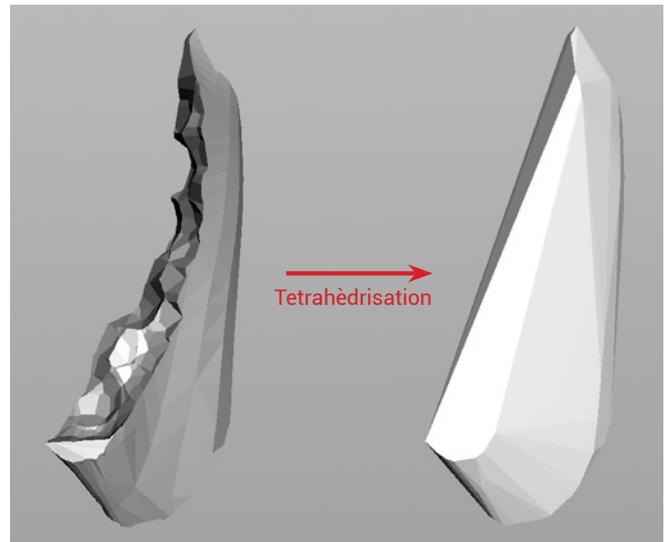
Houdini propose trois moteurs physiques pour la simulation d'interaction entre des objets : Bullet, ODE et RBD.

- **ODE** (Open Dynamics Engine) est un solver open source, créé pour effectuer des simulations sur des géométries primitives, telles que des sphères, des cubes, des cylindres ou des capsules. Il a l'avantage d'être rapide à calculer, mais est en contrepartie très peu précis.
- **RBD** (Rigid bodies Dynamic) est quant à lui extrêmement précis. Il se base sur la forme réelle des géométries en les convertissant en volume. Et est donc beaucoup plus long à simuler.
- La tendance actuelle est d'utiliser **Bullet**, présent depuis la version 12 de Houdini. Ce solver Open source est employé sur de nombreuses grandes productions, telles que 2012, Hancock ou bien Shrek 4. C'est un compromis entre rapidité et précision.
Au sein d'Houdini il tétrahédrise les meshes, donnant ainsi une version simplifiée

de la géométrie en supprimant toutes les formes concaves, ce qui peut causer des problèmes de simulation sur des géométries creuses (ci-contre).

Heureusement j'ai trouvé une parade qui consiste à fracturer l'objet en plusieurs pièces afin que chacune d'elles soit la plus linéaire possible. J'exporte ensuite l'ensemble de ces pièces vers

l'espace de simulation via le Shelf Tool Fractured object afin que Bullet crée une géométrie de collision pour chacun des débris. (ci-dessous).



S'il s'agit d'un objet statique dans la scène, je désactive sa simulation de fracture afin qu'il ne se désintègre pas lors d'un choc ; il sera tout de même perçu comme un mesh de collision.

Si c'est une géométrie qui se déplace je crée des contraintes de glue entre les fractures afin de les rendre solidaires.

Houdini dispose des outils nécessaires pour mesurer la concavité d'une forme, il est donc possible de répartir plus de points de fractures dans les zones creuses afin de créer un mesh de collision précis.

Là où la concavité pose problème, c'est lorsque l'on active l'ajout de détails sur les polygones intérieurs d'une fracture, c'est une option présente dans le node Voronoi Fracture Point qui modifie aléatoirement la position des points

des débris afin de leur donner une forme plus naturelle. Ce qui créé parfois des erreurs de simulation car des objets concaves sont générés.

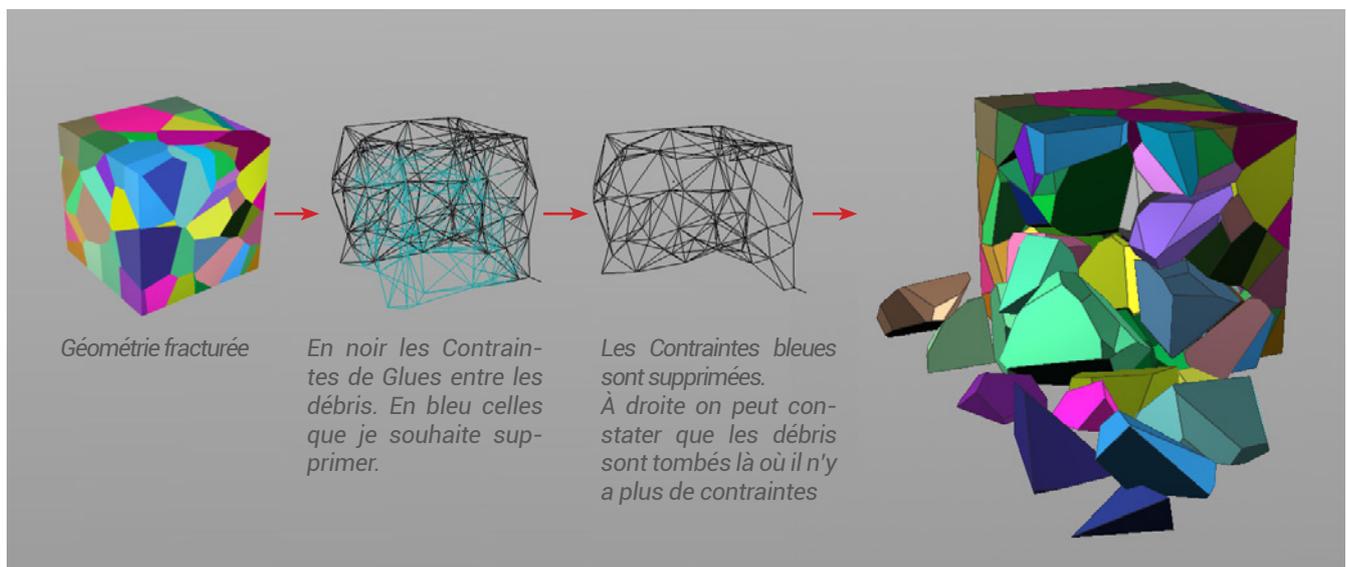
3.1.2.2 Créer des contraintes entre les débris :

Initialement les pièces fracturées se détachent toutes de l'objet dès la première frame. Mais divers moyens existent pour contrôler quand et dans quelle condition elles tomberont.

Le Glue Constraint :

Pour le film, j'ai utilisé le système du Glue Constraint afin de contrôler la manière dont les débris se détachaient du cocon.

Il créé entre chaque pièce un lien de colle matérialisé par une ligne reliant le centre de chacune d'elles. Il est alors possible de le supprimer. Pour ce faire j' ai utilisé une géométrie animée qui appliquait sur le cocon une couleur dans le but de la transférer sur les contraintes, qui grâce à une expression se supprimaient lorsqu'elles étaient peintes. En animant la sphère, je pouvais ainsi contrôler où et quand les pièces se détachaient. Pareillement, j'avais la possibilité de peindre une autre couleur qui serait ignorée par l'expression de destruction, ainsi si une pièce de cette couleur était sur la trajectoire de la sphère elle était ignorée. (Je reviens précisément sur cette technique dans le tutoriel : Fracturer une géométrie, page 70). Grâce à ce principe, j'ai pu créer un système qui permettait aux contraintes situées près du centre de l'objet de ne pas être supprimées. Ainsi, seuls les débris placés à la surface de l'objet se détachaient.

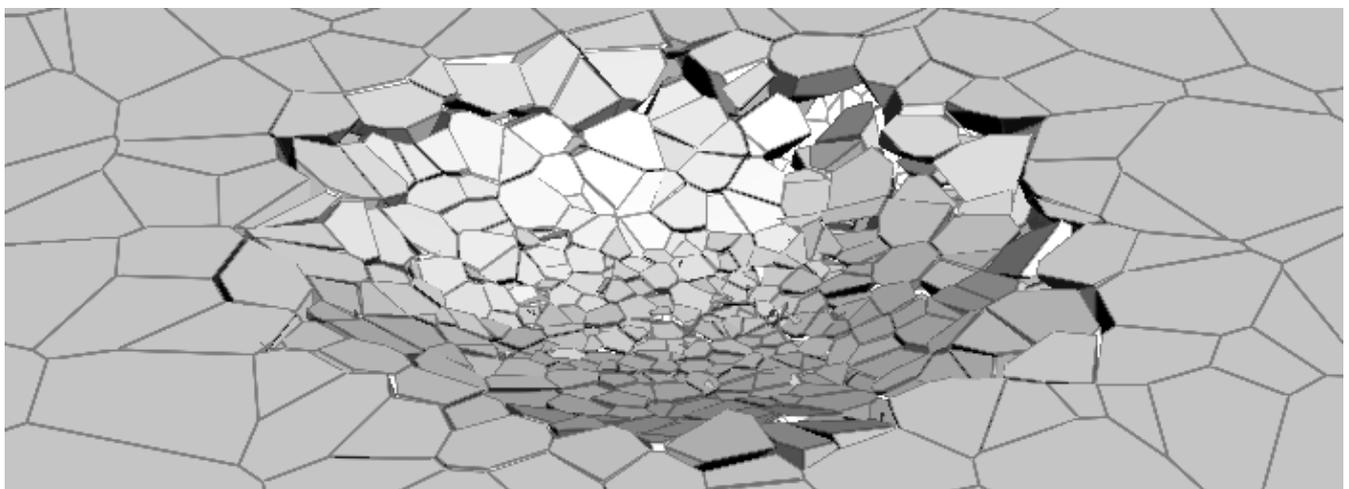
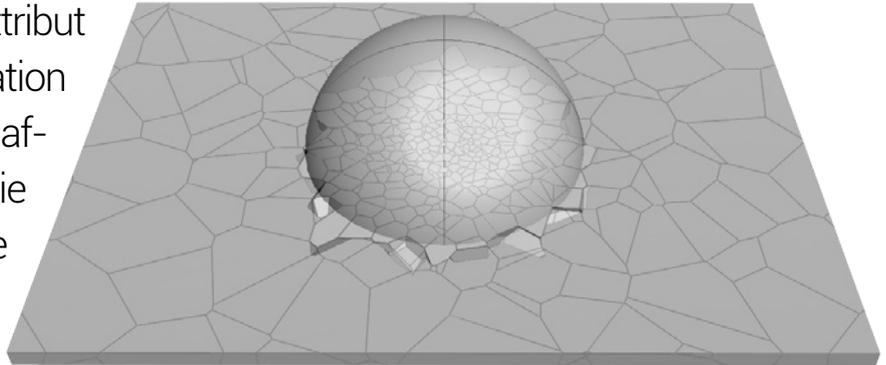


J'avais la possibilité à l'aide d'un ramp similaire à celui utilisé pour contrôler la vitesse des particules, de contrôler précisément jusqu'à quelle profondeur les contraintes pouvaient disparaître.

Node Active Value :

Une autre approche a été d'utiliser au sein de l'espace de simulation le node Active Value. Il permet grâce à une expression de sélectionner très précisément quels éléments de la scène seront simulés ou non. J'ai employé ce système dans mon test vidéo qui représente le sol d'un cachot qui explose :

Une fois mon sol préfracturé, j'avais créé une sphère qui transférait sur les fractures en contact avec elle, un attribut qui allait activer la simulation des pièces afin que le sol s'affaisse. Puis une géométrie invisible, animée sur un axe vertical du sol au plafond, venait collisionner avec ces fractures afin de les projeter dans les airs.



3.2.3 Rendre un plan de destruction :

Le moteur de rendu principal d'Houdini est nommé Mantra. Son architecture basée sur l'algorithme REYES (Renders Everything You Ever Saw), découpe les éléments de la scène en micropolygones plus petits qu'un pixel. Ce principe a été conçu dans le milieu des années 1980 chez LucasFilm,

et est aujourd'hui utilisé dans de nombreux moteurs de rendu dont le célèbre RenderMan de Pixar. De façon habituelle, il ne permet pas de faire usage du Ray Tracing ou de la Global Illumination, c'est pourquoi, il existe deux moteurs Mantra dans Houdini : l'un



Halo 4 Spartan Ops episode 1, par les studios Axis Animation pour Microsoft Games.

est utilisé pour rendre des volumes et des particules de manière optimisée grâce aux micropolygones, et l'autre nommé Mantra-PBR permet de mettre en place des lumières physiquement réalistes.

À l'instar de la suite de court métrage Halo 4 Spartan Ops réalisé par le studio Axis Animation, Mantra peut être utilisé pour créer des images photoréalistes.

Mais conscient que très peu de studios français ont un pipeline adapté à Mantra, j'ai décidé de me mettre en condition et de rendre mon image dans Maya avec Mental Ray. Ça me donnait la possibilité de me confronter à l'exportation d'une géométrie animée depuis Houdini vers un autre logiciel.



Plan du cachot réalisé dans Maya et Houdini.

Pour ce faire, j'ai fait usage du format fbx qui me permettait d'exporter la géométrie animée ainsi que ses UVs.

3.1.3.1 Éclairer un plan de fracture :

Le lighting est une étape importante, elle à d'après moi plusieurs fonctions :

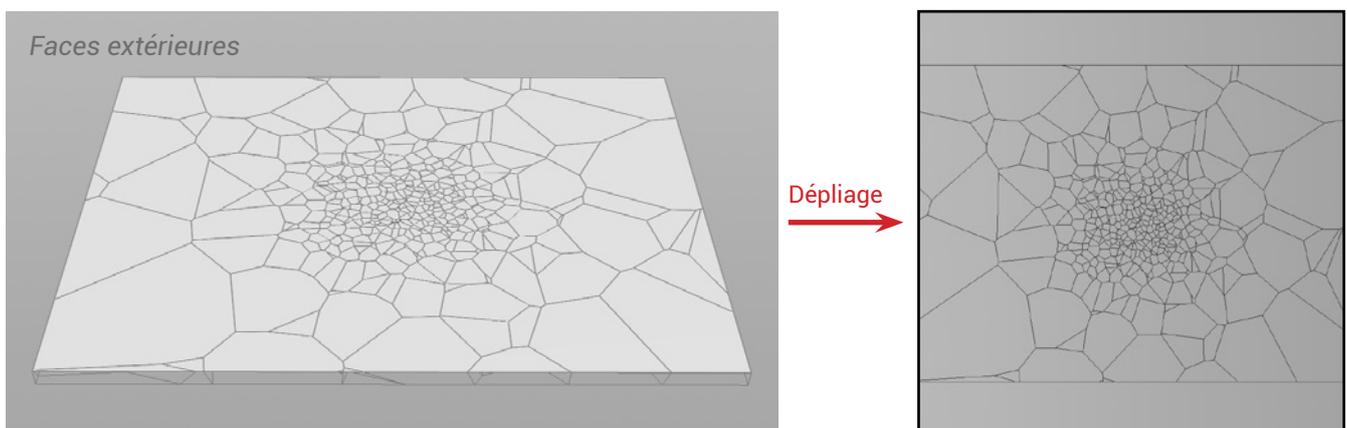
- Donner un ton au plan en créant une ambiance propre au propos que l'on souhaite exprimer.
- Hiérarchiser l'espace, en mettant en valeur les éléments importants dans le but de rendre compréhensibles le décor et son organisation.
- Et donner du volume aux objets.

Ainsi j'ai décidé de créer un éclairage classique constitué d'une zone chaude et d'une zone froide. L'une faisait ressortir l'avant des débris, tandis que l'autre soulignait l'arrière, ce qui permettait de leur procurer un véritable volume.

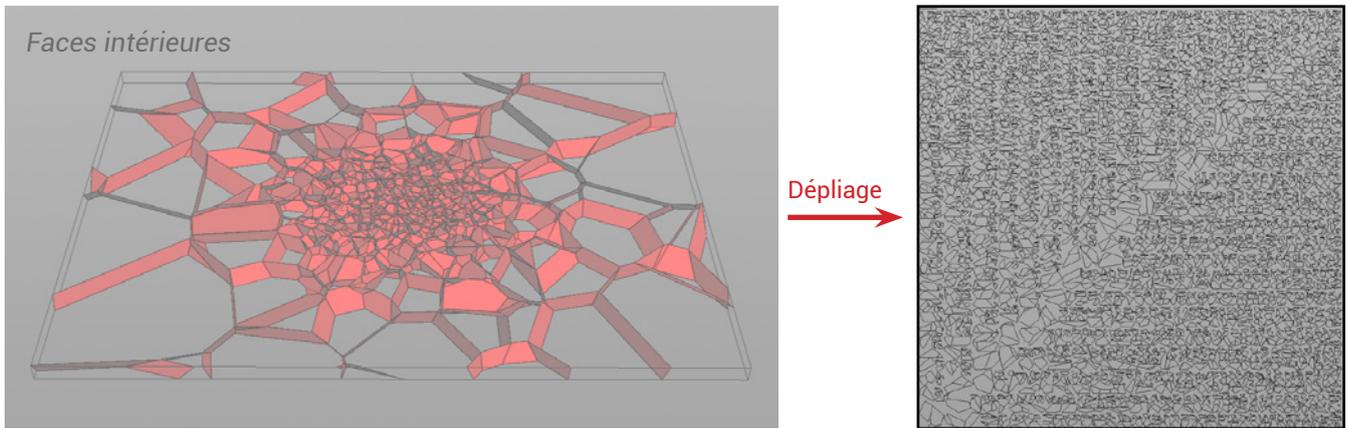
3.1.3.2 Texturer les fractures :

Afin de texturer séparément les polygones situés dans l'épaisseur du sol et ceux en surface j'ai exporté ces deux éléments séparément.

La partie supérieure du sol ne pouvait être texturée que manuellement, car j'avais alors une idée très précise du rendu que je souhaitais obtenir. De plus, le fait que la surface soit parfaitement plane me permettait d'avoir un dépliage Uv qui soit le reflet exact de la géométrie, je pouvais donc la texturer très facilement dans Photoshop.



L'intérieur du sol quant à lui était plus complexe à manipuler. Afin de ne pas perdre de temps sur un dépliage qui respectait la topologie de chaque débris, j'avais décidé d'en créer un de façon automatique. Il séparait, puis disposait les polygones l'un à côté de l'autre dans une map Uv.



Je souhaitais alors les texturer de manière procédurale dans Maya, mais le résultat obtenu était en dessous de mes espérances. J'ai donc décidé d'utiliser Mudbox pour créer les textures. Ce logiciel permet de peindre directement sur une géométrie 3d, puis d'extraire la texture que l'on a réalisée afin de l'appliquer à l'objet dans un autre programme.

Mon problème principal résidait dans la nature même de ma géométrie, qui n'était pas constituée d'une unique pièce, mais d'une centaine, et qui plus est en mouvements. Elles étaient collées les unes aux autres au début de l'animation, puis s'éparpillaient dans le décor.

Afin d'être en mesure de peindre l'ensemble des débris, j'ai donc exporté la géométrie depuis Maya vers Mudbox à une image précise où les fragments étaient assez espacés pour être texturés.

Puis une fois la texture terminée, je l'ai appliquée à l'ensemble du mesh animé.

Contrairement à ce que l'on pourrait croire, ce workflow m'aura offert la possibilité de créer les textures en moins d'une heure, c'est en partie dû à la passerelle qui existe entre Mudbox et Maya qui permet d'importer et d'exporter des géométries texturées en un simple clic.

3.1.3.3 Ajout de détail :

Par souci de réalisme il me fallait ajouter de la poussière projetée par l'explosion. J'ai donc généré un fluide qui prenait comme source d'émission certains des plus gros débris. Dans un premier temps, j'ai mis en place une première version dans Houdini, mais comme je voulais me replonger dans

les fluides Maya, ma version finale a été réalisée avec ce dernier. J'ai ensuite agrémenté mes fumées de stock-shots afin d'ajouter plus de détails. Puis j'ai advecté quelques particules à la fumée pour créer de la poussière fine. L'ensemble de ces éléments donnait à la scène beaucoup plus de vie, grâce aux nombreuses interactions qui se créaient entre les différents objets.

Pour conclure, deux méthodes ont été utilisées pour créer cet effet :

- Une manuelle (le node paint),
- Et une procédurale (utilisation de géométries primitives pour transférer un attribut ou répartir des points).

Cette dernière permet de nombreuses déclinaisons telles que générer des fractures ou détacher des pièces au contact de particule ou de fumée.

D'après des expériences passées, Houdini semble pouvoir m'offrir plus de possibilités que certains plugins de destructions disponibles sur des logiciels courants tels que Maya ou 3DS Max, ainsi je souhaiterais poursuivre mes recherches dans ce domaine, et créer une scène totalement autonome et capable de fracturer d'elle-même un ensemble de géométrie suivant certaines interactions.

3.2 DÉGRADER UNE MEMBRANE :

Le film montrait un ensemble de plans très serrés de forêt, il fallait donc mettre en place de nombreux détails pour rendre le film réaliste. Ainsi il était au départ question d'une membrane, qui, placée autour du papillon, devait se déchirer lorsque le cocon se fracturait. C'était une piste secondaire que nous voulions exploiter dans le cas où nous aurions eu le temps durant la production. Dans cette éventualité, j'avais donc engagé de petites recherches pour ne pas être pris au dépourvu.

3.2.1 Dans Houdini :

Il n'était malheureusement pas possible de créer une simulation de cloth pour cet effet, car l'animation du papillon demandait beaucoup de temps, et n'aurait pas été disponible assez tôt, durant les trois semaines du projet,

pour que je puisse la paramétrer et la simuler. C'est alors que j'ai découvert le node spring, qui donne aux edges un comportement similaire à celui d'un ressort, et permet aux points de réagir à diverses forces telles que la gravité ou le vent. Il est très optimisé en raison de l'absence de self-collision, c'est pourquoi ses polygones ont tendance à s'interpénétrer sous l'influence des forces que l'on applique à la géométrie. Il peut cependant réagir de manière correcte aux collisions avec d'autres géométries.

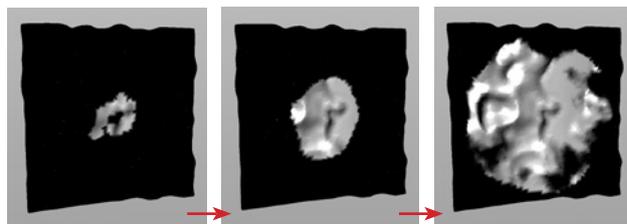
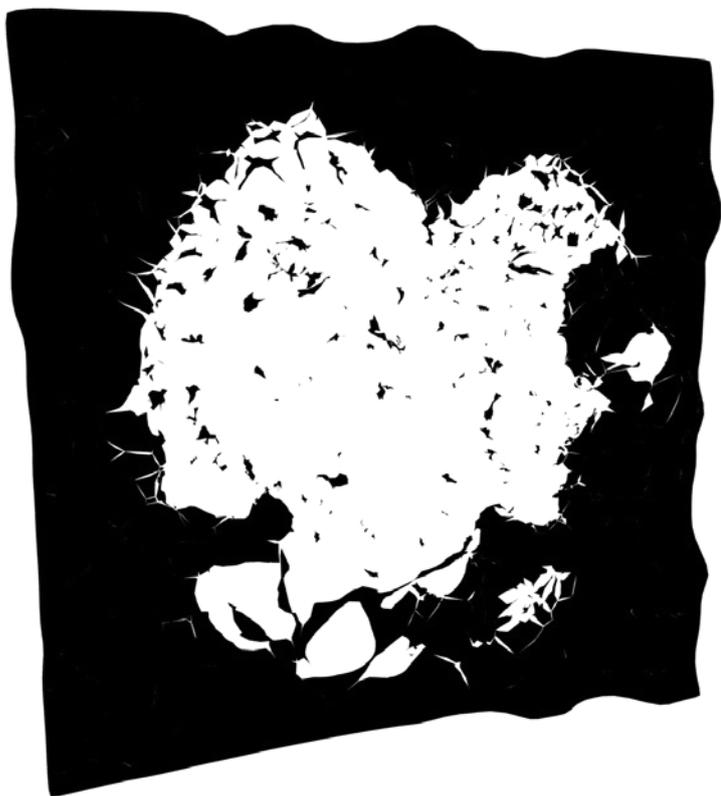
À noter que ce node n'établit pas de simulation à proprement parler, car il fonctionne au sein du contexte Géométrie.

Mon but avec le Spring était de donner à la membrane un comportement similaire à celui d'une surface qui se consume. Ce qui consistait à en détacher au fur et à mesure de la combustion des morceaux, puis à les faire se replier sur eux-mêmes.

Le node spring, grâce à un paramètre de tension, permettait précisément de créer cet effet de torsion, en rapprochant chaque point qui compose l'objet les uns des autres.

Il ne me restait alors plus qu'à fracturer la géométrie, puis à créer une texture animée qui allait sélectionner des points dans le but de constituer un groupe, qui, chargé dans le paramètre Fixed Point du node, allaient être ignorés par

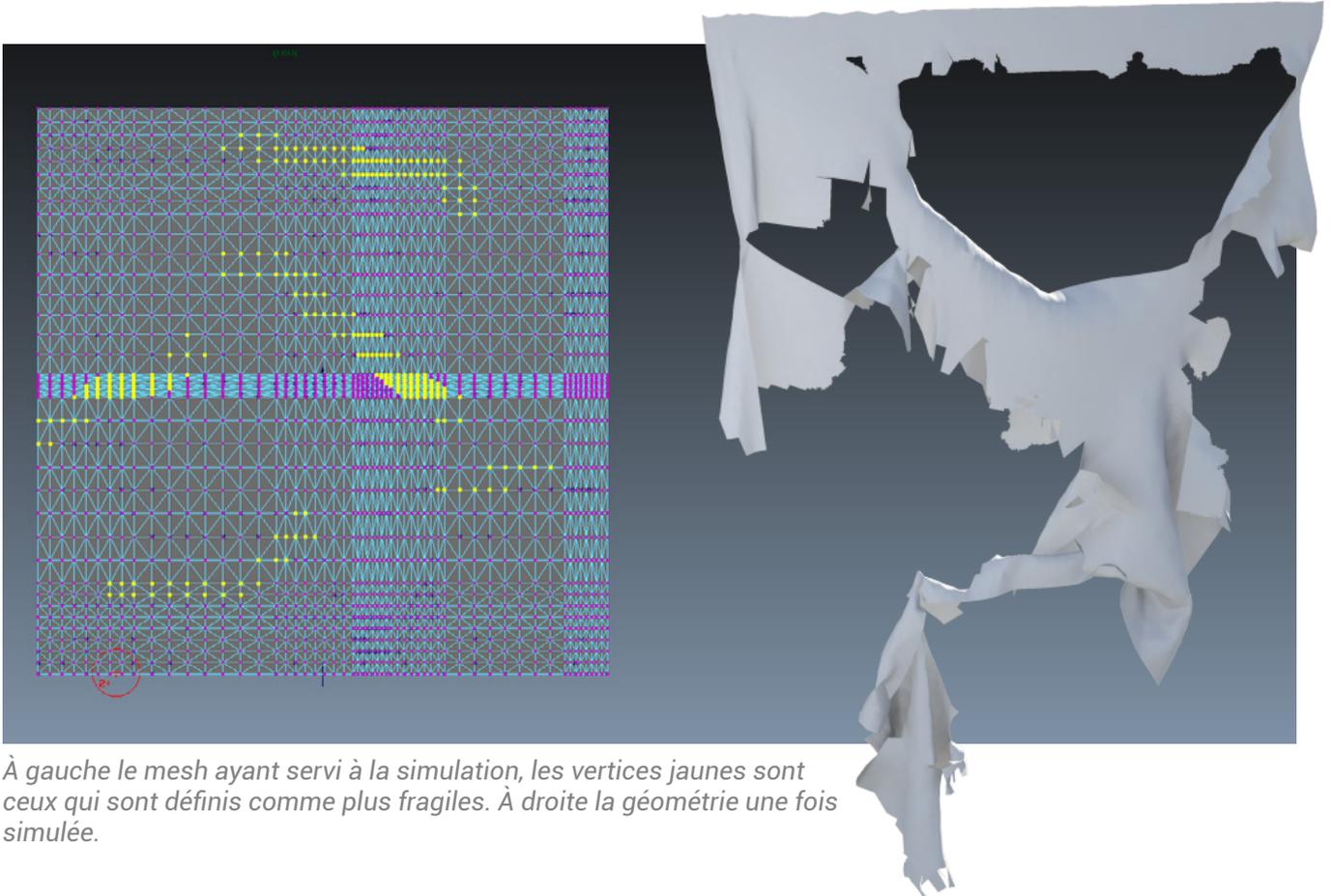
l'effet. Ainsi j'avais la possibilité d'activer la torsion des fragments dans l'espace et le temps.



À gauche la géométrie qui se décompose grâce au node Spring. Ci-dessus progression de la texture, les points noirs seront ignorés du node Spring.

3.2.2 Dans Maya :

J'avais entamé parallèlement des recherches de cloth sur Maya. Elles consistaient à rendre une géométrie plus fragile dans certaines zones. Ainsi lors de la simulation, le cloth se déchirait là où certains vertices avaient été définis comme tearable (déchirable).



À gauche le mesh ayant servi à la simulation, les vertices jaunes sont ceux qui sont définis comme plus fragiles. À droite la géométrie une fois simulée.

De la même manière qu'avec Houdini, il est possible dans Maya de peindre des attributs sur un cloth (la masse, la rigidité, la résistance, etc.).

Le temps de simulation est assez similaire sur les deux logiciels, mais habitué à l'approche non destructive de Houdini qui me permettait d'appliquer un effet rapidement à un nouveau mesh, j'ai très vite mis de côté mes recherches sur Maya.

CONCLUSION :

Bien qu'il n'existe pas de méthode particulière pour créer et contrôler des effets spéciaux, et ce quel que soit le domaine (fluides, particules, rigides bodies, etc.), j'ai pu dégager au cours de l'année l'utilisation de deux principes récurrents :

- Le premier laisse au logiciel le contrôle sur certaines actions, tel que dans mon test pour dégrader une membrane, où la texture se répandait d'elle-même à la surface de l'objet, puis supprimait les polygones colorés.

Ce qui donne à l'effet une certaine autonomie. Il est alors capable de créer de lui-même des visuels auxquels on n'aurait pas pensé.

- Le second quant à lui, offre une supervision totale à l'artiste grâce à la création de contrôleurs personnalisés qui permettent de diriger l'effet précisément, ainsi pour mon test de particules, de nombreuses courbes me donnaient la possibilité de modifier leurs comportements au cours du temps.

Il s'agit alors de mettre en place un outil qui au gré des réglages offre différentes déclinaisons de l'effet.

Ma méthode de travail s'est davantage tournée vers une approche procédurale qui était plus souple qu'une approche manuelle destructive. Je pouvais ainsi réutiliser d'un simple clic un effet sur des géométries différentes.

À l'heure actuelle, les logiciels 3D sont assez évolués et accessibles pour permettre à tout infographiste de créer des simulations complexes. Sur l'ensemble des programmes que j'ai eu l'occasion d'utiliser, Houdini me semble le plus adapté à la création d'outils pour les effets spéciaux, car il donne la possibilité de créer tous types d'effets sans demander de solides connaissances en programmation, bien que ce soit nécessaire pour une utilisation avancée.

Suite à cette année de recherches, je souhaiterais pousser plus en profondeur l'interaction entre les différents domaines (particules, fluides et rigid bodies), afin d'être en mesure de créer des plans complexes alliant contrôles manuels et procéduraux.

Thèses :

PETER CLAES, 2009. *Controlling Fluid Simulation with Custom Fields in Houdini*. National Centre for Computer Animation Bournemouth University.

Disponible à l'adresse : http://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/MSc11/Ahmad/aghourab_MSc_Thesis_Public.pdf

AHMAD GHOURAB, 2011. *A Fluid Implicit Particle Approach to a Pyro Solver in Houdini*. National Centre for Computer Animation Bournemouth University.

Disponible à l'adresse : http://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/MSc11/Ahmad/aghourab_MSc_Thesis_Public.pdf

RAVINDRA DWIVEDI, 2011. *Art Directable Tornadoes*. Office of Graduate Studies of Texas A&M University.

Disponible à l'adresse : <http://repository.tamu.edu/bitstream/handle/1969.1/ETD-TAMU-2011-05-9417/DWIVEDI-THESIS.pdf?sequence=2>

EMANUELE GOFFREDO, 2010. *A Tool for Procedural Destruction in Houdini*. National Centre for Computer Animation Bournemouth University.

Disponible à l'adresse : http://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/MSc2010/04EmanuelleGoffredo/Thesis_Emanuele_Goffredo.pdf

ANDY ABGOTTSPON, 2011. *Houdini Cracking Tool*. National Centre for Computer Animation Bournemouth University.

Disponible à l'adresse : <http://www.aaweb.ch/wp-content/portfolio/bu/CGI-Techniques-Project.pdf>

Publications :

MAGNUS WRENNINGE & NAFEES BIN ZAFAR, 2011. *Production Volume Rendering Fundamentals*. SIGGRAPH 2011

Disponible à l'adresse : <http://magnuswrenninge.com/content/pubs/ProductionVolumeRenderingFundamentals2011.pdf>

RYOICHI ANDO, NILS THÛREY & CHRIS WOJTAN, 2013. *Highly Adaptive Simulations on Tetrahedral Meshes*. SIGGRAPH 2013.

Disponible à l'adresse : http://pub.ist.ac.at/group_wojtan/projects/2013_Ando_HALSoTM/download/tetflip.pdf

Cours en ligne :

PETER QUINT, PQHoudini, Houdini Tutorials Channel :

<https://vimeo.com/channels/54102>

Site de SideFX, : http://www.sidefx.com/index.php?option=com_content&task=blog-category&id=192&Itemid=351

Tutoriaux :

ANDREU LUCIO , *Houdini Fractal Clouds & Terrain*, CMIVFX.

ADAM SWAAB . *Houdini Connections*, CMIVFX.

Houdini Dust And Smoke Technique For Visual Effects TD's, CMIVFX.

GEORGES NAKHLE . *Houdini Intro To Particle Animation*, CMIVFX.

BRIAN GOODWIN. *Houdini Fluid Effects For TD's*, CMIVFX.

ADAM SWAAB . *Houdini 3D Galaxy Creation Effects*, CMIVFX.

ALVARO SEGURA . *Houdini Network Relationships Vol1*, CMIVFX.

ALVARO CASTAÑEDA, , *Houdini Hip Tricks Volume 1, 2, 3, 4*, CMIVFX.

En préambule à cette année de recherche, j'avais décidé de me constituer une station de travail personnalisée dans le but de disposer d'une puissance de calcul et d'une réactivité non négligeable pour les nombreuses simulations à venir.

Pour les VFX, il est essentiel d'avoir une bonne capacité de stockage, du fait de la place que peuvent prendre les dossiers de cache, et particulièrement lorsqu'il s'agit d'une simulation s'étalant sur un grand nombre d'images. En effet, plus une image est placée loin dans l'animation et plus il y a d'informations à écrire en cache. Un cache consiste à enregistrer sur le disque dur des informations issues de la simulation de sorte que le logiciel lors de la prochaine lecture n'ait pas à effectuer de nouveau le calcul. Pour plus d'efficacité lors de l'écriture et de la lecture, il faut donc opter pour un disque dur ayant une rapidité d'écriture importante.

De même, la mémoire vive n'est pas à négliger, d'une part pour être en mesure d'afficher des scènes volumineuses, mais aussi pour avoir la possibilité de lancer des simulations complexes sans que le logiciel sature par manque de place.

La plupart des nodes d'Houdini supportent le multithread, il utilise donc de manière optimale l'ensemble des coeurs. Ainsi on bénéficie du maximum de puissance de calcul que le processeur peut fournir.

La carte graphique quant à elle n'a pas besoin d'être extrêmement puissante, bien que le logiciel supporte pour certains nodes le calcul GPU c'est le CPU qui effectue l'essentiel des calculs à l'heure actuelle.

Nombre d'utilisateurs Houdini se sont regroupés au fur et à mesure des années en une communauté qui n'hésite pas à partager ses connaissances sur les forums ou par le biais de tutoriaux. Avec cette annexe j'apporte aussi ma contribution en espérant à mon tour aider d'autres personnes.

J'y aborde de manière technique la création de certains effets développés au cours de cette année. Au-delà d'une simple liste d'actions à effectuer, ces tutoriaux grâce à de nombreuses notes et commentaires vous éclairons sur le fonctionnement du logiciel et de ses nodes.

Voici la liste des tutoriaux :

- **Fumée** : Peindre une zone de chaleur sur un mesh 1/2
Le node paint : 45

- **Fumée** : Peindre une zone de chaleur sur un mesh 2/2
Le node attribut Transfer : 54

- **Particules** : diriger des particules le long d'une curve
Le point cloud : 57

- **Fracture** : préfracturer & simuler
Le Glue Constraint : 70

- **SOP** : Répandre depuis un point d'origine une couleur sur la surface d'un mesh
Le point Cloud : 88

FUMÉE : PEINDRE UNE ZONE DE CHALEUR SUR UN MESH 1/2 LE NODE PAINT. (H 12)

Une fumée monte lorsqu'elle est chaude, et descend lorsqu'elle est froide.
Notre but est de faire interagir une fumée avec des zones de chaleur, afin de la faire s'élever.

Fichier joint : Temperature.hip

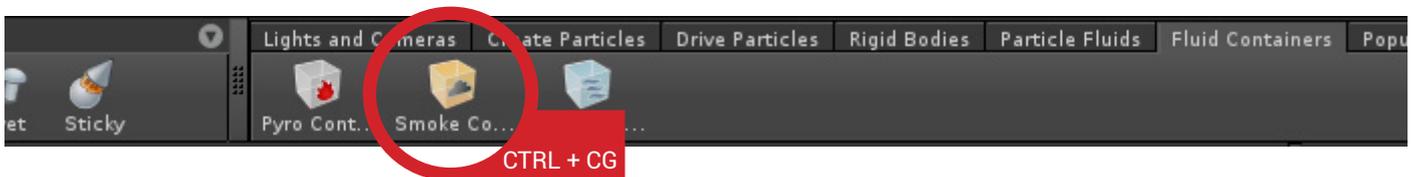
MISE EN PLACE DE LA SCÈNE :

Émettre de la fumée:

Dans le shelf *Creat* faire un **Ctrl + Clic-Gauche** sur l'option *Sphere*.



Dans le shelf *Fluid Containers* faire un **Ctrl + Clic-Gauche** sur le shelf tool *Smoke Container*.



Via le *Network Editor* retourner dans la vue contextuelle **Scene**.

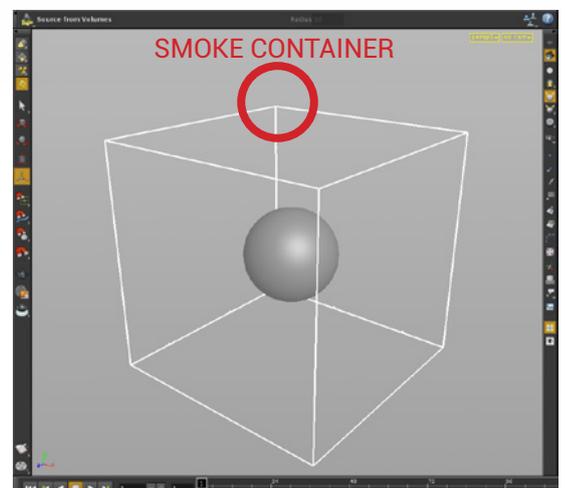
Sélectionner le node *sphere_object1*.

Dans le shelf *Populate Containers* faire un **Alt+Clic-Gauche** sur l'option *Source from Volume*.



Puis dans la *Scene View* sélectionner le *Smoke Container*, et presser la touche **ENTRER**.

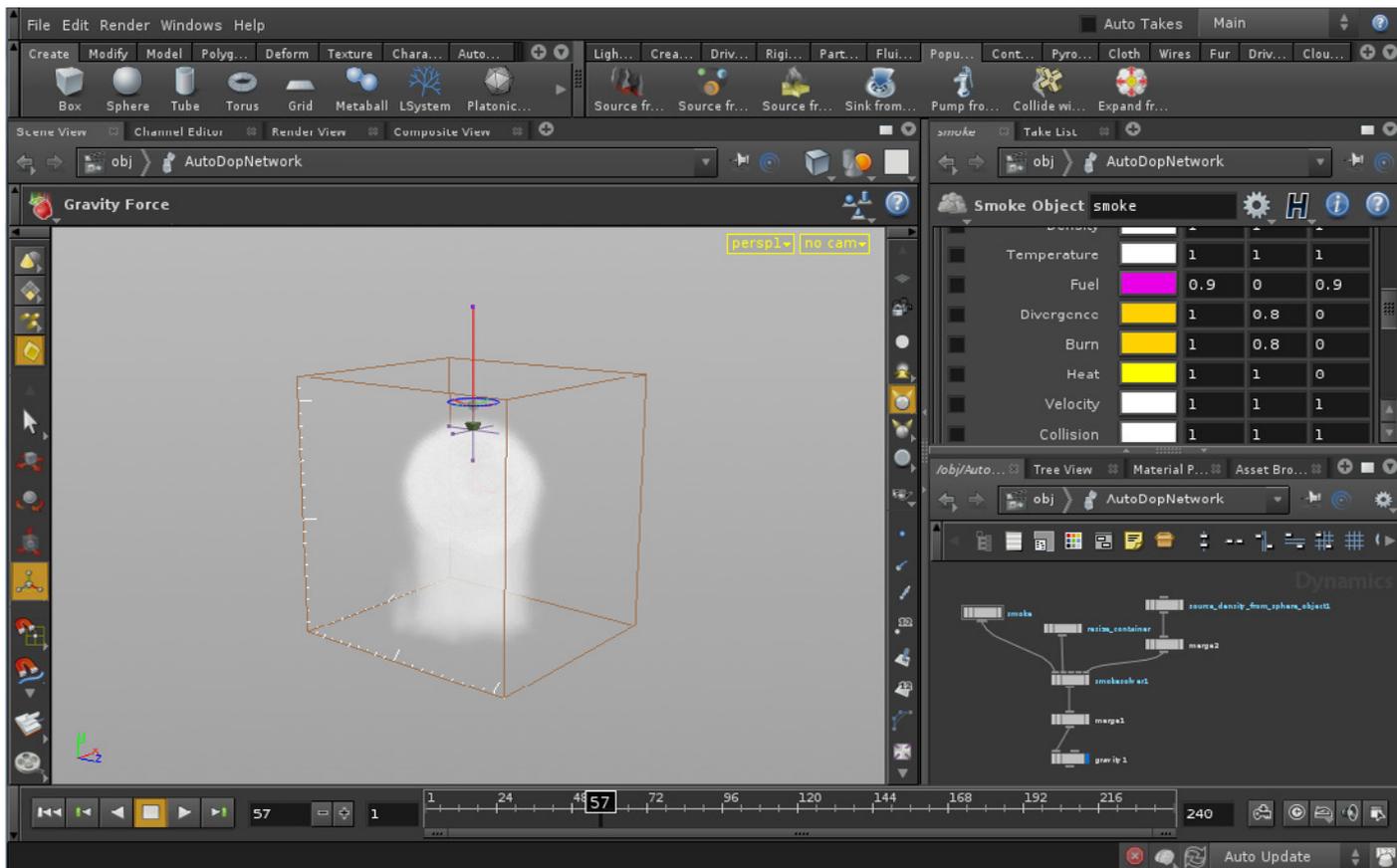
Le shelf tool *Source From Volume* va convertir une géométrie en source de fumée.
Le *smoke container* sert d'espace de simulation, la fumée ne pourra être émise au-dehors.



La sphère émet à présent de la fumée.

Retourner dans la vue contextuelle **Scene**, puis rentrer dans le node *AutoDopNetwork*.

Presser la touche L pour réorganiser l'espace.



Dans la scène view, chaque cran blanc autour du Smoke Container correspond à une unité de la grille de voxel.

Les voxels sont les éléments qui constituent un fluide. Ils peuvent être comparés à des pixels 3d et sont capables de contenir de nombreuses informations. Plus la densité de la grille est élevée et plus la fumée sera définie.

Ajuster la taille de l'espace de simulation :

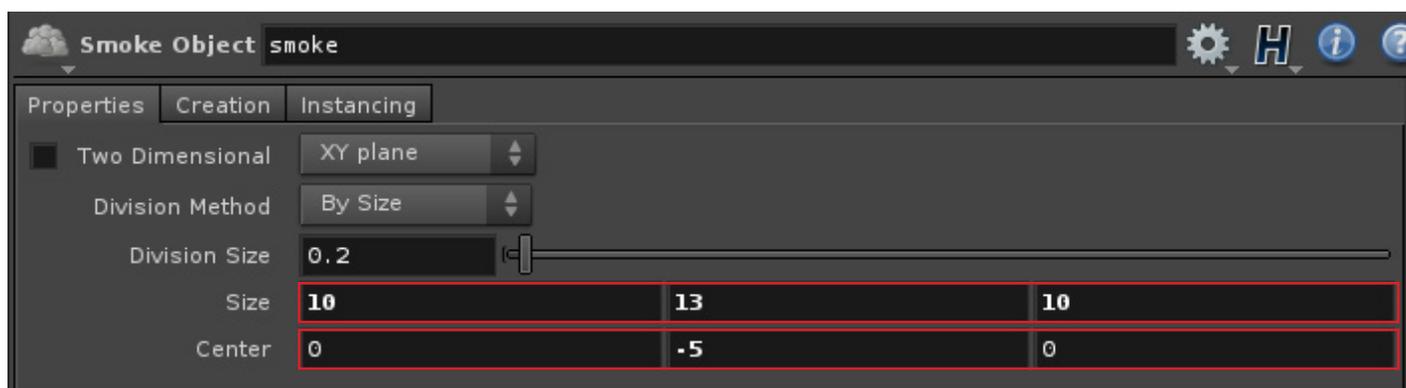
Sélectionner le node *pyro*.

Dans l'onglet **Propriétés** mettre comme valeurs au paramètre *Size* :

1	0	1	3	1	0
---	---	---	---	---	---

au paramètre *Center* :

0	-5	0
---	----	---



Créer le sol :

Dans le shelf **Create**, faire un **CTRL + Clic-Gauche** sur l'option *Box*.

Dans le **Network Editor**, double-cliquer sur le node *box_object1*.

Une fois à l'intérieur, appuyer sur la touche **TAB**, puis écrire : **Transform**

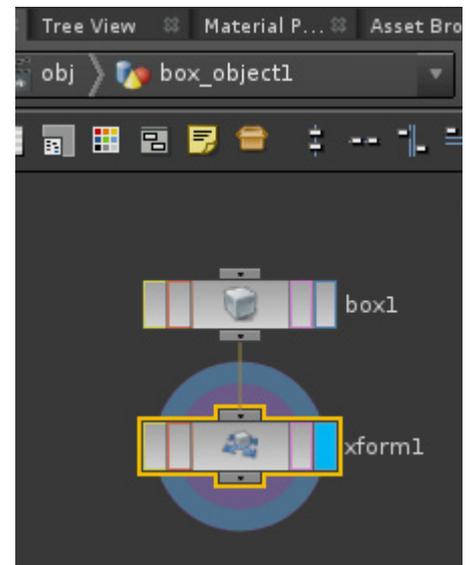
Relier l'Output du node *box1* à l'Input du node *xform1*.

Le node *Transform* sert à modifier la position, l'orientation ou l'échelle d'un objet.

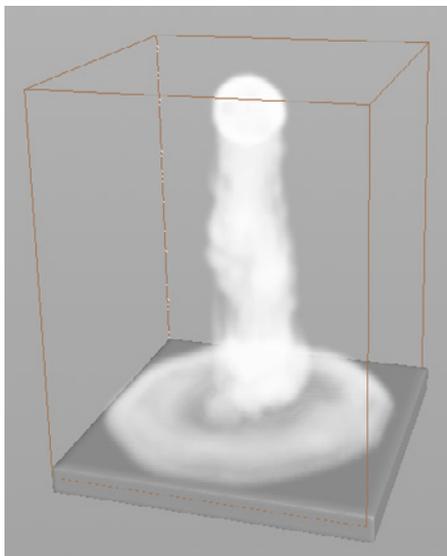
Sélectionner le node *xform1*.

Mettre aux valeurs du paramètre *Translate* : **0 -11 0**
Scale : **10 1 10**

Ainsi le sol correspond aux dimensions du conteneur de fumée.



Créer une collision entre le sol et la fumée :



Retourner dans la vue contextuelle **Scene**. Puis sélectionner le node *box_object1*.

Dans le shelf *Populate Containers* faire un **ALT + Clic-Gauche** sur l'icône *Collide with Objects*.



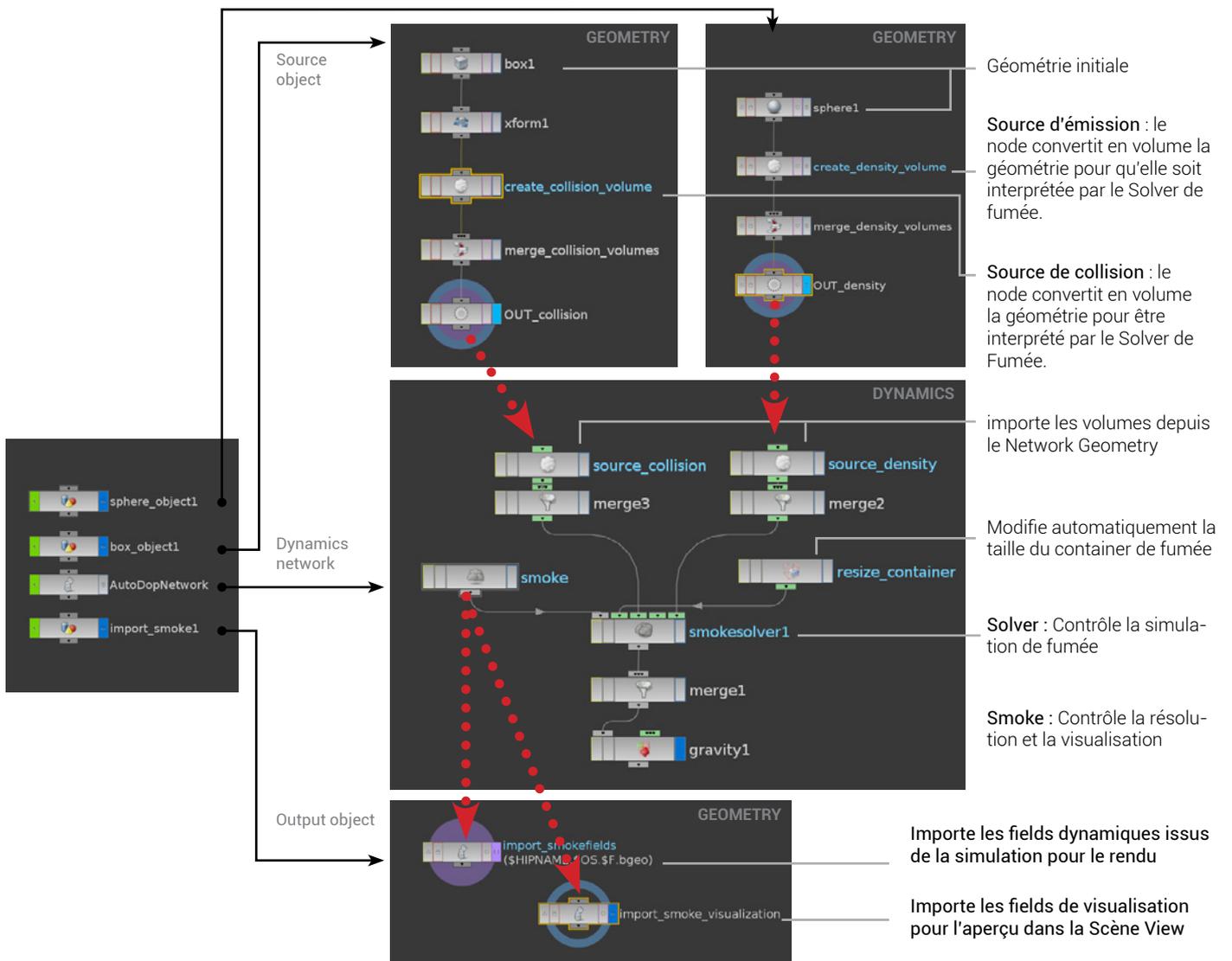
Sélectionner le *Smoke Container* et appuyer sur la touche **ENTRER**.

Si on retourne dans l'*AutoDopNetwork* et qu'on lance la simulation on peut voir que la fumée entre en collision avec la box. Le node *gravity* permet à la fumée de descendre.

NOTE :

Houdini est organisé en plusieurs couches telles que : Scène, Géométries, Dynamique, Texture, etc. Chaque couche à une fonction précise, et peut communiquer avec les autres. Ainsi nous créons l'émetteur et le sol dans la couche Géométrie, cette dernière sert à modifier les géométries. La simulation quant à elle s'effectue dans la couche Dynamics grâce au Smoke Solver. Pour que ce dernier soit en mesure d'importer et d'interpréter des géométries, il faut qu'elles soient transformées en volume. À l'instar de la fumée, les volumes sont constitués de voxels, ils ont l'avantage d'indiquer précisément dans l'espace 3D, quelle est la valeur de tel ou tel attribut (densité, température, collision, etc.). Ainsi le volume issu de la sphère va indiquer au solveur de fumée, quelle est la densité de l'émetteur et sa température.

Illustration du fonctionnement de la scène :



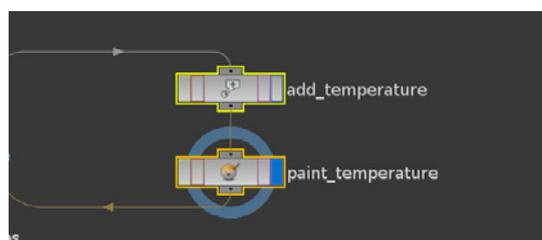
PEINDRE DES ZONES DE CHALEUR :

Préparer les connexions :

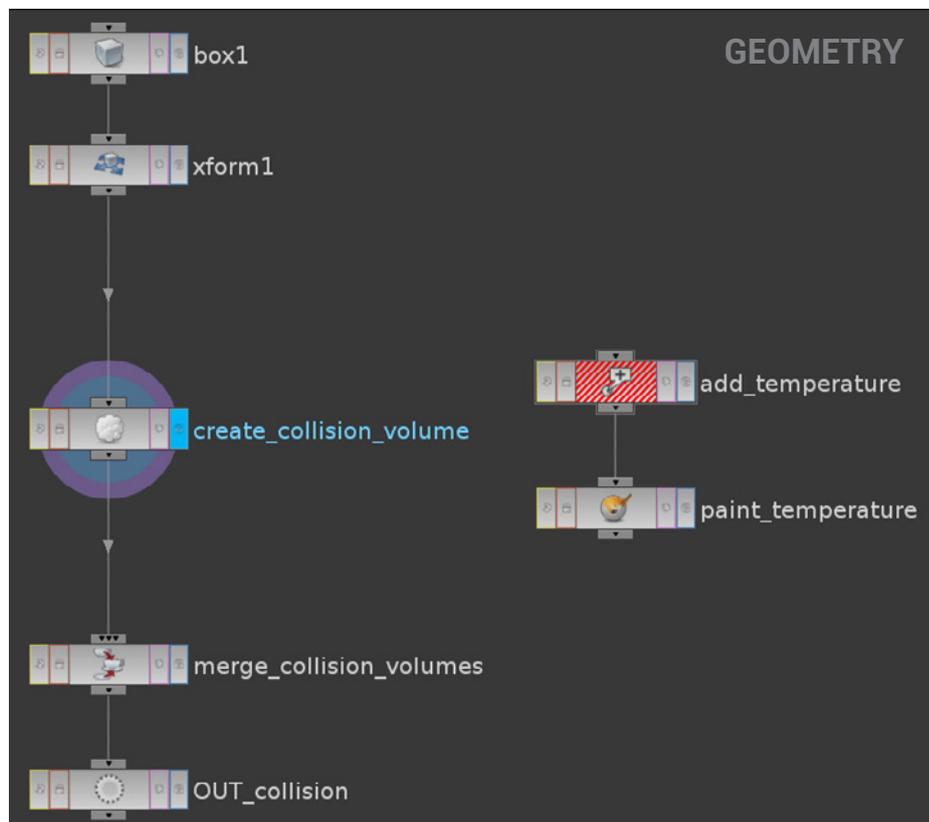
Retourner dans la vue contextuelle **Scene**. Puis sélectionner le node *box_object1*. Dans le shelf *Containers Tools*, faire un **ALT + Clic-Gauche** sur l'icône *Paint Temperature*.



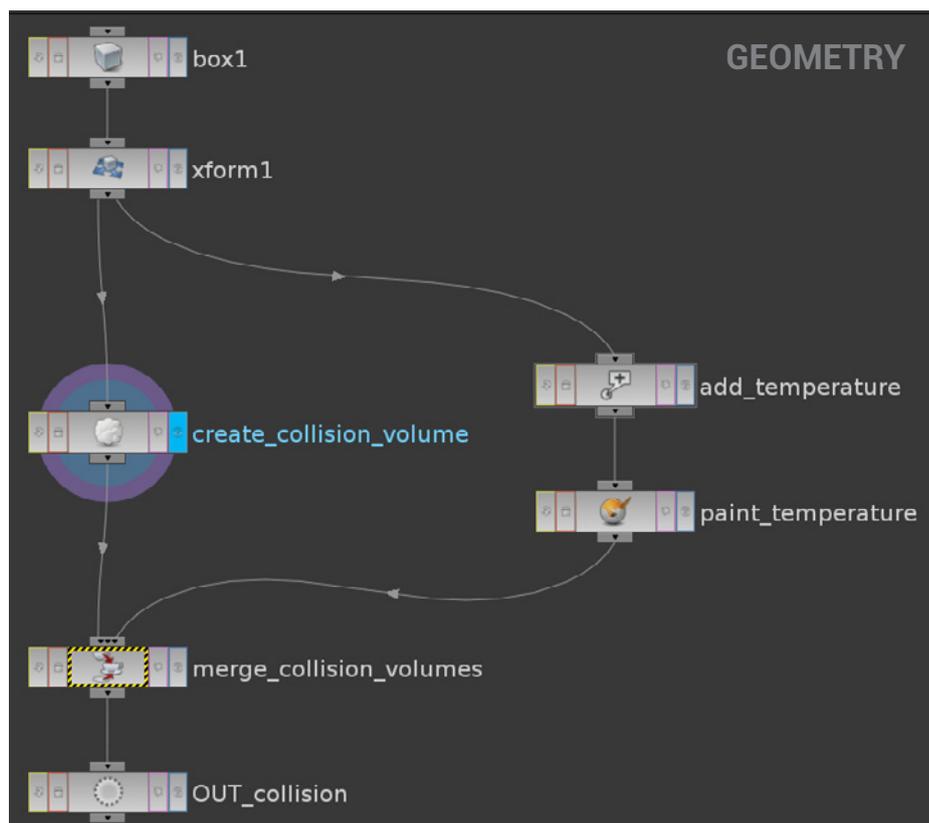
Dans le node *box_object1*, deux nouveaux nodes se sont ajoutés :



Les sélectionner, puis cliquer sur l'un d'entre eux, et tout en restant appuyé, secouer la souris de droite à gauche énergiquement afin de les détacher du reste de l'arbre. Puis organiser l'espace de travail comme suit :



Enfin relier le Input du node *add_temperature* au Output du node *xform1*, et relier le Output du node *paint_temperature* à l'Input du node *merge_collision_volume*.

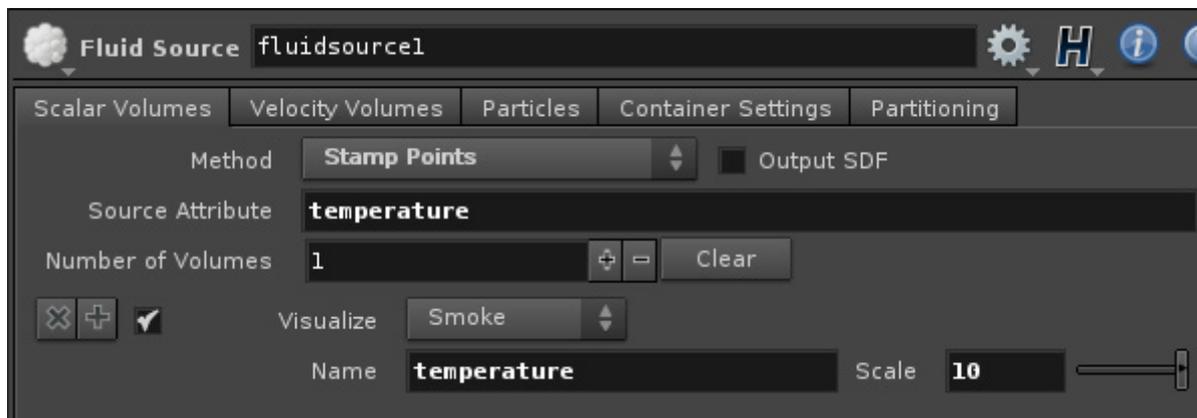


Explication :

Ces deux nodes auraient pu être créés manuellement comme l'ensemble des Shelves Tools. Il s'agit d'un node **Attribut Creat** (le *Add_temperature*), qui a pour fonction de créer un attribut **temperature** de classe **Point**, et d'un node **Paint** (le *paint_temperature*), qui sert à peindre un attribut sur une géométrie, ici la température. Le node *paint* applique une couleur sur les points de la géométrie c'est pourquoi l'attribut de température est de class Point.

Ajouter un node *FluidSource* entre le node *paint_temperature* et le node *merge_collision_volume*. Dans le node *fluidsource1* à l'onglet *Scalar Volume*, mettre le paramètre *Method* sur : **Stamp Points**.

Puis écrire dans les paramètres *Source Attribut* et *Name* : **temperature**. Enfin augmenter le *scale* à 10.



Dans le node *creat_collision_volume* aller dans l'onglet *Scalar Volume*, puis dans le sous-onglet *SDF From Geometry*, enfin désactiver la case : *Scale By Source Attribut*.

Explication :

Le node *fluidSource* va créer un *Volume* de type **Scalar** d'après l'attribut spécifié dans le paramètre *Source Attribut*. Ce volume transmettra les informations de température au solveur de fumée.

Ce qui se trouve dans le paramètre *Source Attribut* doit être de class **Point**, car nous utilisons la méthode *Stamp Point* pour générer le volume.

Le paramètre *Scale* nous sert à régler l'intensité de la température.

Note :

Il y a deux types de volume : les **Scalar** et les **Vector**.

Le **Scalar** écrit sur chaque voxel des informations de *type float*, alors que le **Vector** se sert de *vecteurs*.

Les informations de couleur issues du node *paint* une fois transformées en attribut *temperature* sont de *type float*, nous utilisons donc un **Scalar volume**.

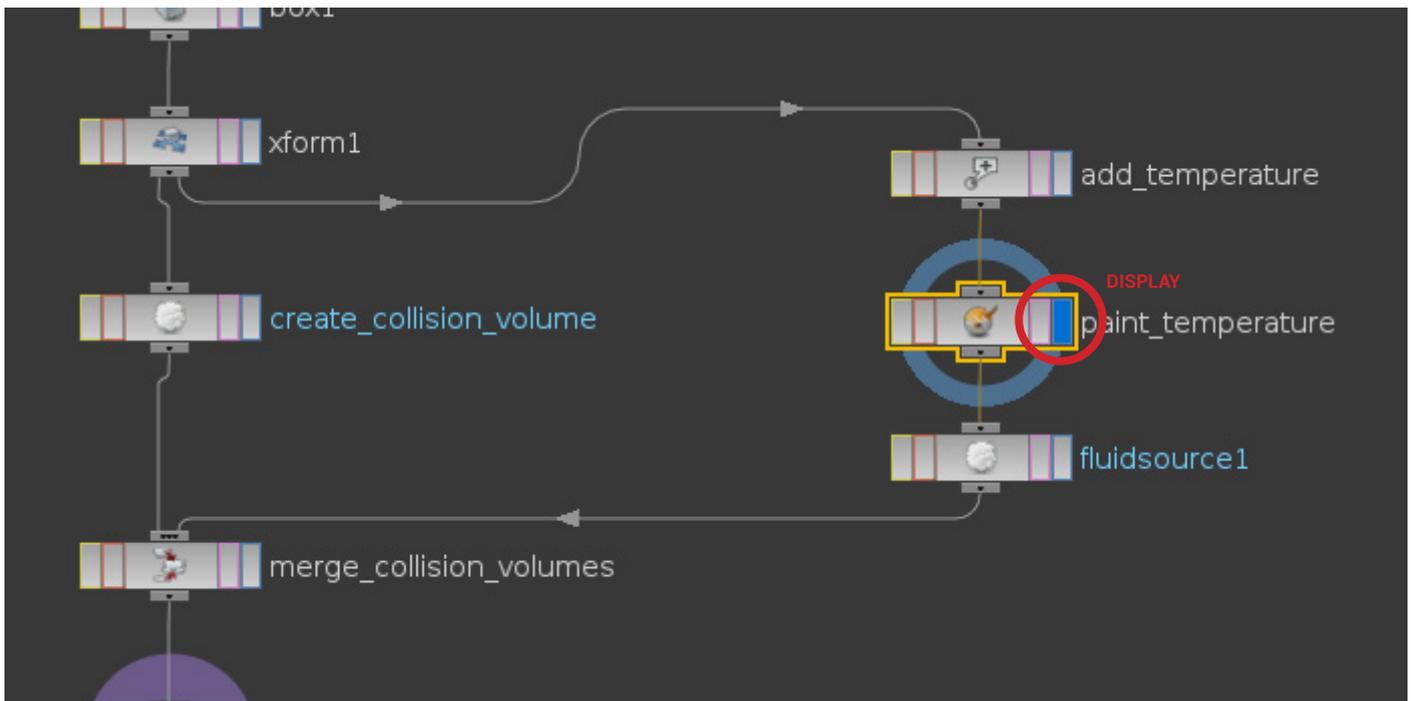
Peindre la température :

Afin de pouvoir peindre la température sur les points de la géométrie, nous devons lui ajouter des divisions :

Sélectionner le node *box1*, mettre son *Primitive Type* sur *Polygon Mesh* et augmenter son *axis division* comme suit : **60 60 60**



Sélectionner le node *paint_temperature*, puis activer son Display pour pouvoir peindre.

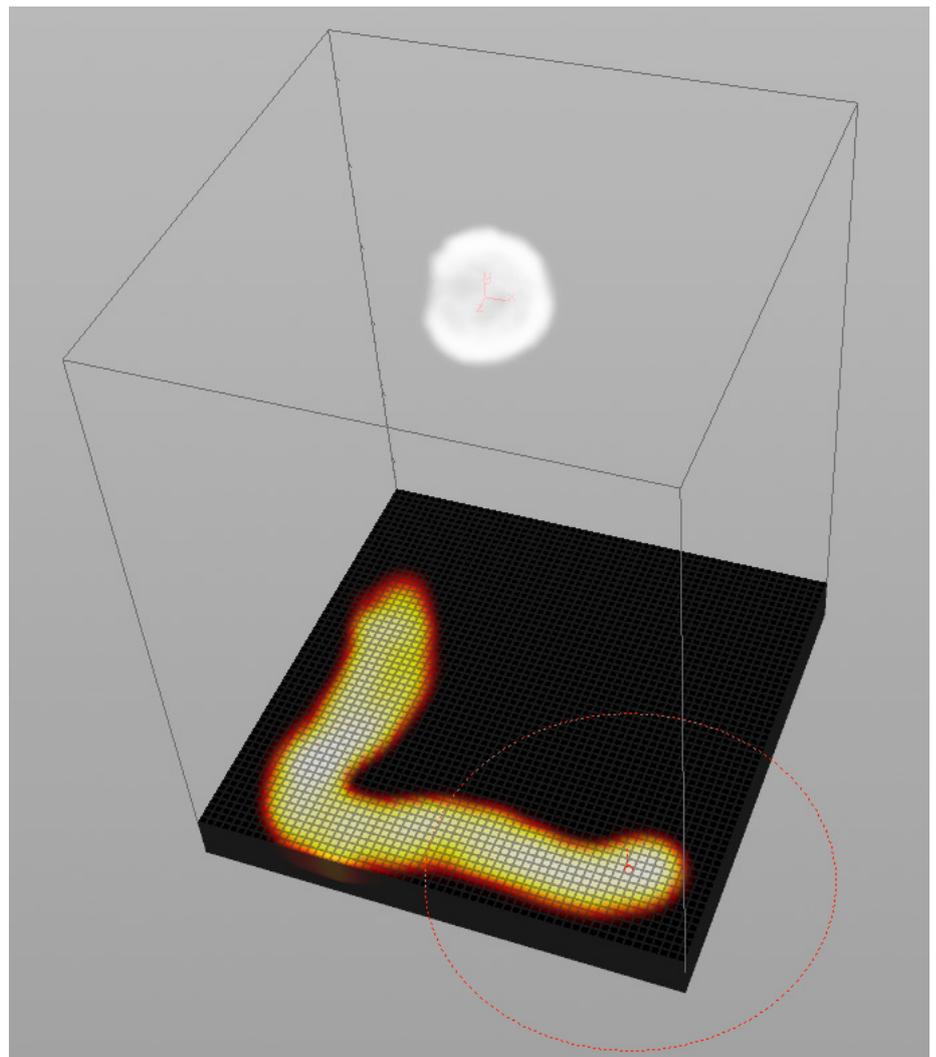


La box deviendra alors noir.

Afin de rendre le pinceau plus large maintenir **SHIFT + Clic- Gauche** et faire glisser la souris vers la droite.

Pour peindre, utiliser le **Clic- Gauche** de la souris, et employer le **Clic du milieu** pour effacer.

L'onglet Brush du node *paint_temperature* permet de modifier les caractéristiques du pinceau (opacité, taille, pression, etc.)



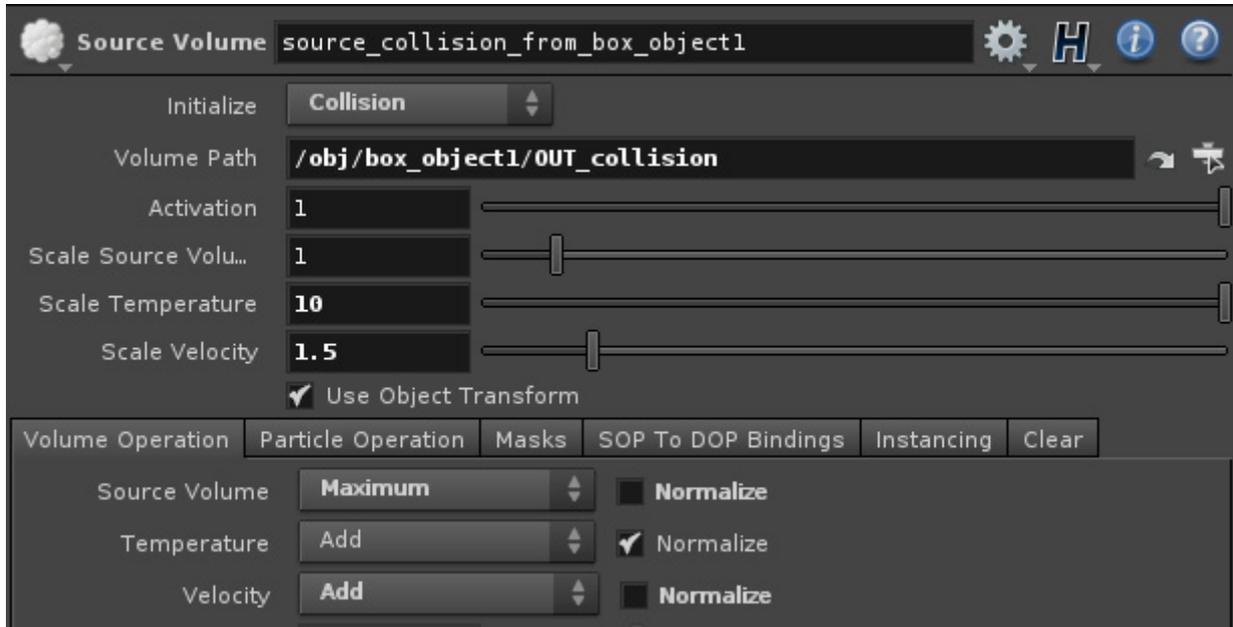
Paramétrer l'AutodopNetwork :

Retourner dans la vue contextuelle **Scene**. Puis rentrer dans le node *AutoDopNetwork*.

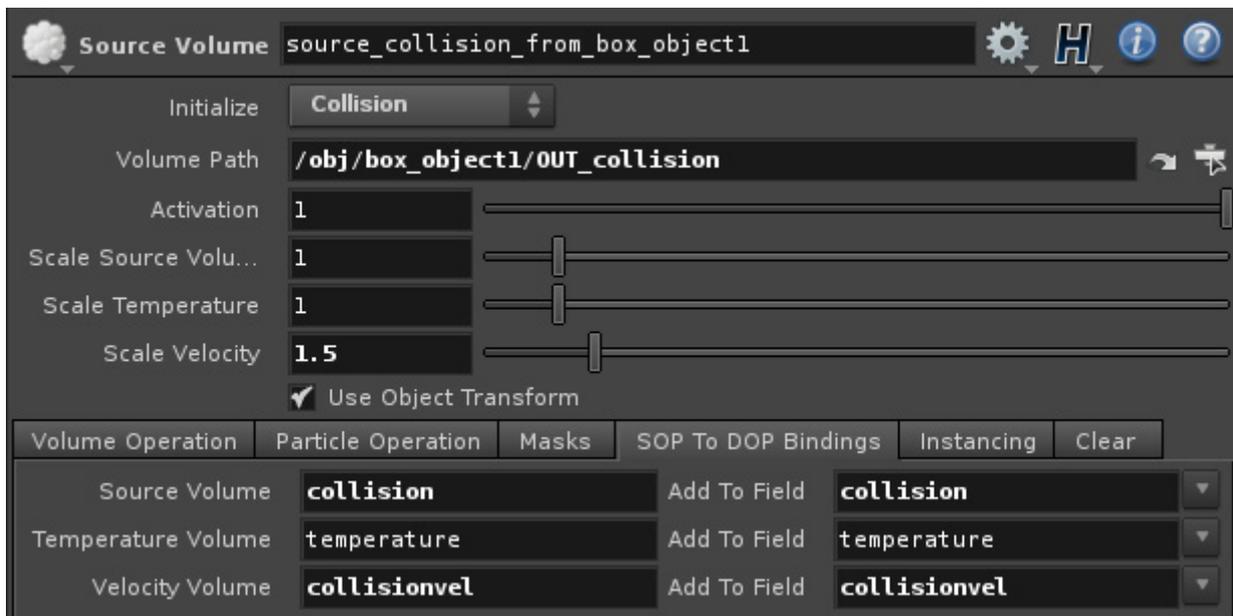
Sélectionner le node *source_collision_from_object1*.

Augmenter le paramètre *Scale Temperature* à **10**.

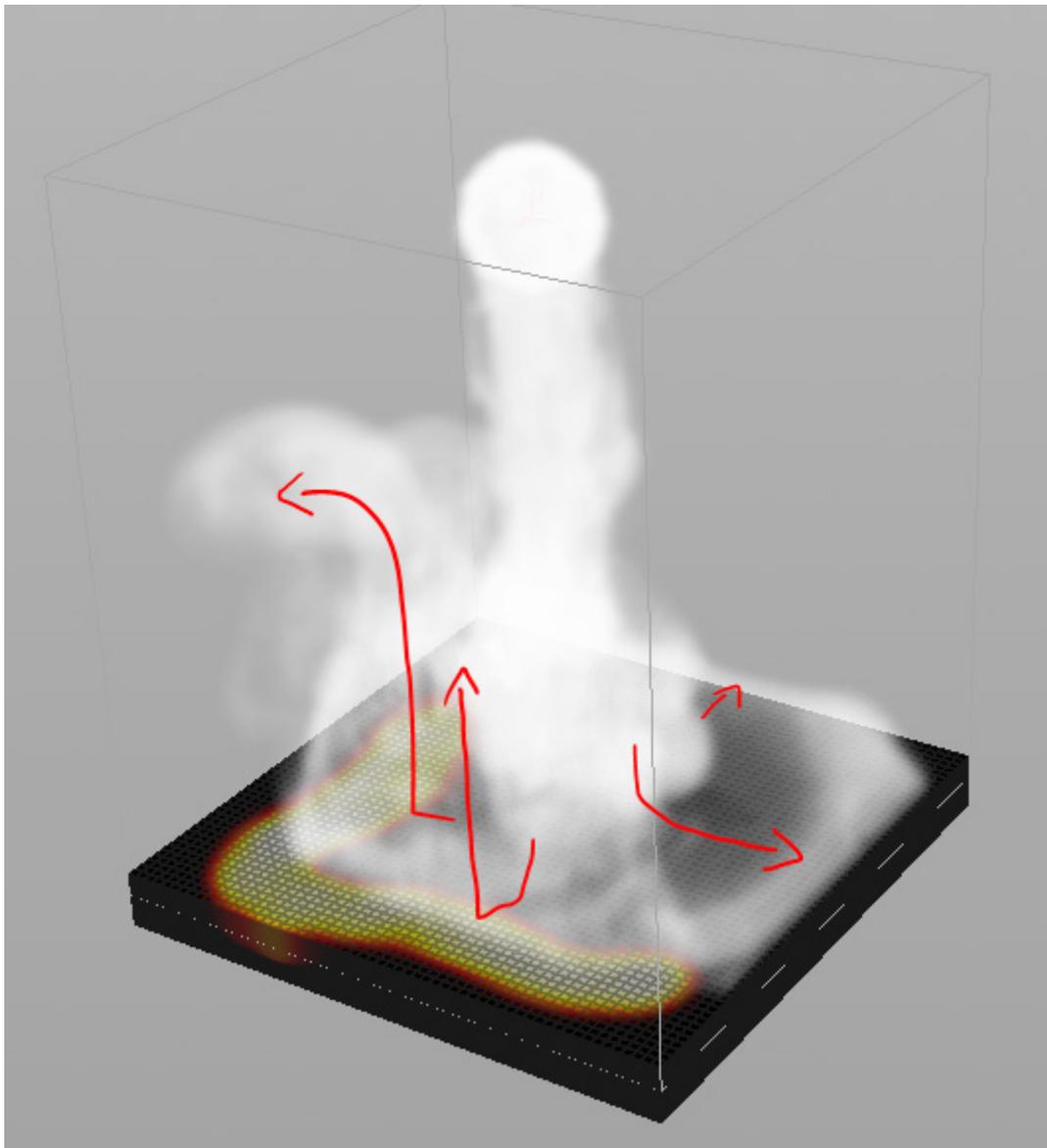
Dans l'onglet *Volume Operation*, mettre le paramètre *Temperature* sur : **Add**.



Puis dans l'onglet *SOP To DOP Binding*, écrire : **temperature** dans les deux champs vides du paramètre *Temperature Volume*.



Ainsi le node *Source Volume* ira chercher au sein du SOP *box_object1*, l'attribut *temperature* (transformé sous forme de volume par le node *Fluid Source*), afin de l'ajouter à son Field *temperature*.



Si les zones de chaleur n'ont pas d'impact sur la fumée, augmenter le paramètre **scale** qui se trouve dans le node *box_object1 > fluidsourc1*.

Ainsi que le paramètre **Scale temperature** du node *source_collision_from_box_objec1* de l'AutoDopNetwork.

Note :

Dans l'AutoDopNetwork, le node nommé *smokesolver1* est un **Solver**. Les Solvers ont pour rôle de calculer le comportement des éléments simulés (RBD, CLoth, Fluid, etc.). Ses différents inputs lui apportent des informations physiques issues d'un ou plusieurs objets, et de son environnement, tel que la densité, la gravité, la vitesse, etc.

Notre solver de fumée a besoin de connaître la source d'émission du fluide, ainsi que la zone de chaleur. C'est ce qu'apportent les nodes *source_collision_from_box_objec1* et *source_density_from_sphere_objec1*, on appelle ces nodes des **Microsolvers**.

Ils vont chercher dans les SOPs des volumes dans lesquels sont inscrites au sein de leurs voxels des informations spécifiques (dans notre cas la température et la densité). Ces informations sont ensuite interprétées par le Source Volume et envoyées au Solver principal.

Les quatre Inputs verts du *smokesolver* permettent d'interpréter les *microsolvers* à des instants différents : avant ou après la simulation.

FUMÉE : PEINDRE UNE ZONE DE CHALEUR SUR UN MESH 2/2 LE NODE ATTRIBUT TRANSFER. (H 12)

Nous allons nous servir d'une sphère animée pour peindre les zones de chaleur.

Fichier joint : Temperature_transfere.hip

PRÉPARATION DE LA SCÈNE :

Reprendre la scène **Temperature.hip**.

Via le *Network Editor* retourner dans la vue contextuelle **Scene**.

Rentrer dans le node *box_object1*.

Supprimer les nodes *add_temperature* et *paint_temperature*.

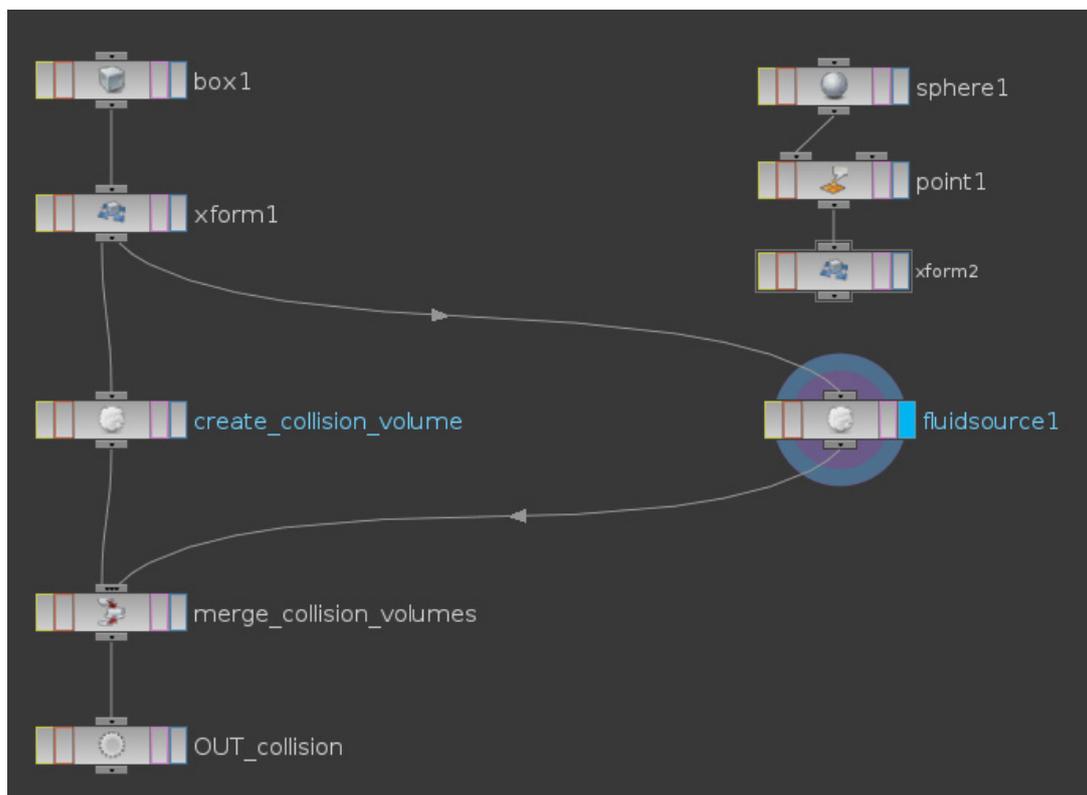
MISE EN PLACE DE L'EFFET :

Nous allons faire en sorte qu'une sphère rouge transmette sa couleur sur le sol en se déplaçant. Un node *Attribut Creat* appliquera un attribut **temperature** sur ces zones rouges. Puis le node *fluid source1* créera un volume d'après cet attribut, et l'enverra au solver de fumée via un node *Source Volume*.

Au sein du node *box_object1*, créer un node *sphere*.

Créer un node *point*, puis relier l'Output du node *sphere1* à son **premier Input**.

Créer un node *transform*, puis le relier à l'Output du node *point1*.



Dans le node *point*, mettre le paramètre *Keep Color* sur *Add color*, faire un **Clic-Droit** sur *Color* puis sélectionner *Delet Channels*. Enfin mettre ses valeurs RVB à : 1 0 0

Notre sphère est à présent rouge.

Afin que le channel rouge de la sphère soit le seul à avoir des valeurs de 1, nous devons rendre la box noir.

Créer un node *Point* entre les nodes *xform1* et *fluidsource1*. Suivre la même procédure que pour le node *Point* précédent, et mettre ses valeurs RVB à : 0 0 0

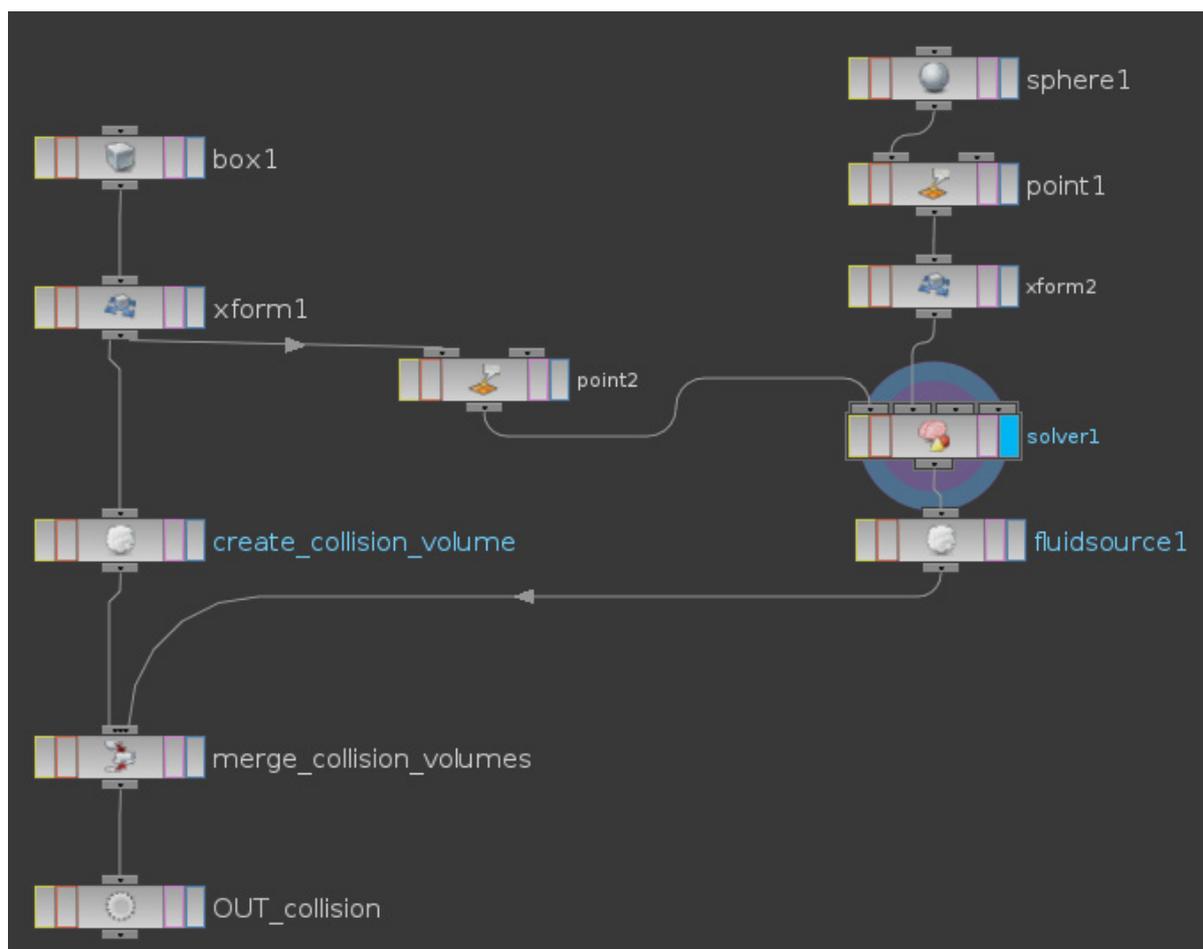
Faire en sorte que la sphère applique sa couleur sur la box en se déplaçant :

Pour que la sphère transmette sa couleur, nous allons utiliser un node *Attribut Transfer*. Nous souhaitons que les points qu'elle colore restent rouges après son passage. Nous allons donc créer un node *solver*, qui non seulement contiendra les connections d'attribut transfer, mais qui en plus, tâchera de superposer le résultat de la frame actuelle à la précédente, permettant donc de garder le sol coloré.

Mettre les valeurs XYZ du paramètre *Translate* du node *xform2* à : -3.4 -10.4 -3.4

Créer un node *Solver*. Le placer entre le node *point2* et le node *fluidsource1*.

Mettre dans son second Input le node *xform2*.



Rentrer dans le node *solver1*.
Créer un node *Attribut Trans-*
*fer*¹.

Brancher le node *Prev_Frame*
à son premier Input. Puis le
node *Input_2* à son second
Input.

Dans le node *attributtrans-*
fer1, aller à l'onglet **Con-**
ditions. Baisser le Distance
Threshold à : 1.

Ainsi le rouge² se répandra sur
la box dans un rayon sem-
blable au volume de la sphère.

Retourner à l'extérieur du
solver.

Puis ajouter un node *Attribut*
*Creat*³ entre les nodes *solver1*
et *fluidsource1*.

Mettre dans son paramètre
Name : **temperature**

Puis à son paramètre *Value* :
\$SCR

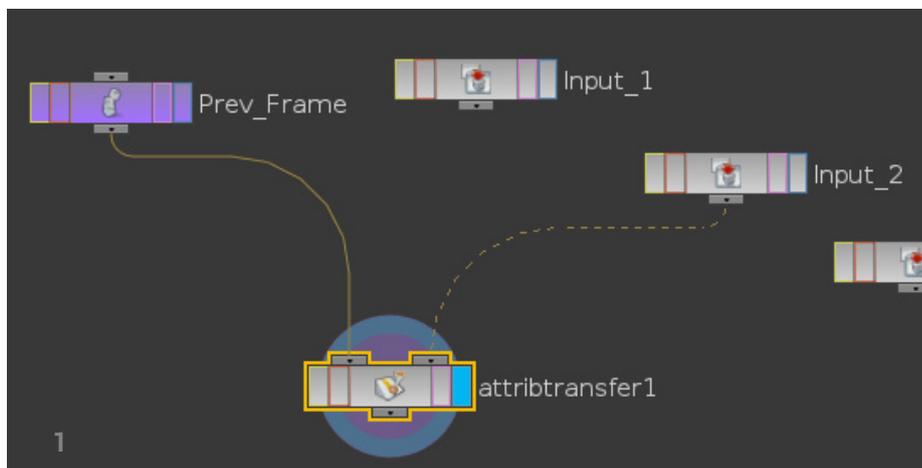
Animer la sphère :

Sur le node *xform2* faire un **Alt**
+ Clic-Gauche sur le paramètre
Translate afin de créer des
images clefs.

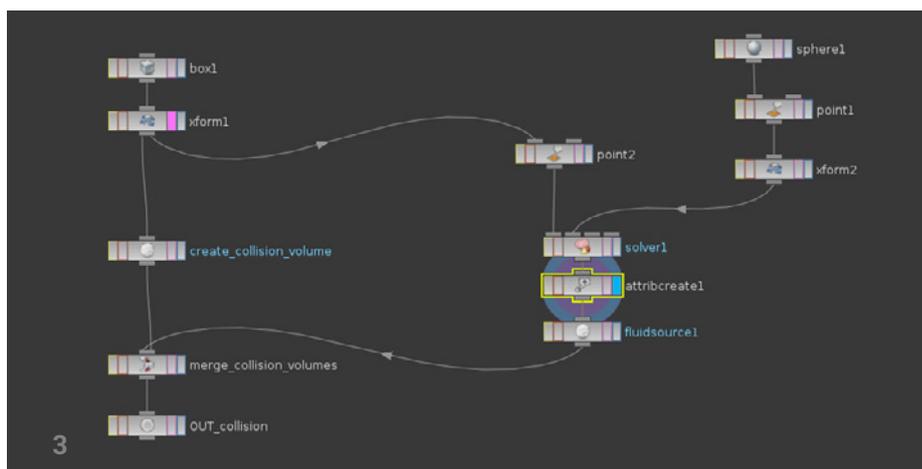
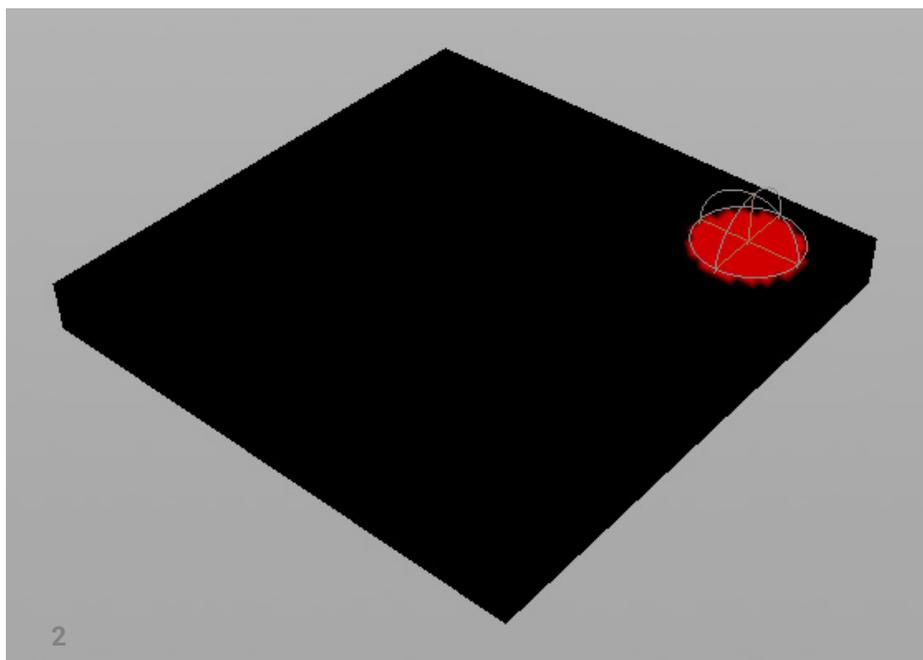
Augmenter la résolution de la fumée :

Dans l'*AutoDopNetwork* sélec-
tionner le node *smoke*, puis
baisser la valeur du paramètre
Division Size afin d'avoir des
voxels plus petits et donc une
meilleure résolution.

Pour augmenter la taille de la
zone de simulation, utiliser le
paramètre *Size*.



L'input_2 correspond au second input du node *solver*, c'est-à-dire à celui relié à la sphère.



Note :

\$SCR correspond au channel de couleur rouge.

Le node créera donc un attribut *temperature* de class *point* se basant sur les valeurs du channel rouge de la box.

PARTICULES : DIRIGER DES PARTICULES LE LONG D'UNE CURVE POINT CLOUD (H 12)

Dans ce tutoriel, nous allons diriger des particules le long d'une curve avec l'utilisation du Point Cloud.

Fichier joint : Particles.hip

LA CURVE :

Nous souhaitons donner à la curve une forme de spirale. Pour ce faire, nous allons utiliser les fonctions mathématiques cos et sin que nous implémenterons sous forme d'expression dans le paramètre position d'un node Point.

Créer un node Géométrie dans la vue contextuelle *Scene*.

Le renommer : **CURVE**.

Double-cliquer sur le node afin de rentrer à l'intérieur.

Supprimer le *file1*, et créer un node *Line*.

Mettre son paramètre *Distance* à : **15**.

Puis le paramètre *Points* à : **1000**.

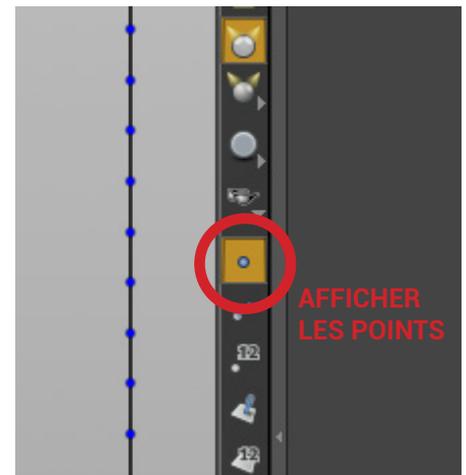
Brancher un node *Point* en dessous du *line1*.

Écrire dans le channel X du paramètre *Position* :

$\cos(\$PT * 3) * \$PT / 80$

Écrire dans le channel Y :

$\$PT / 30$



Pour visualiser les points, activer le display à droite de la scène view.

Note :

Les channels X et Z vont mettre en forme la spirale, le channel Y va la scaller le tout sur un axe vertical.

\$PT : Cette variable sera remplacée alternativement par le numéro de chaque point qui compose la curve, et ce une fois par image.

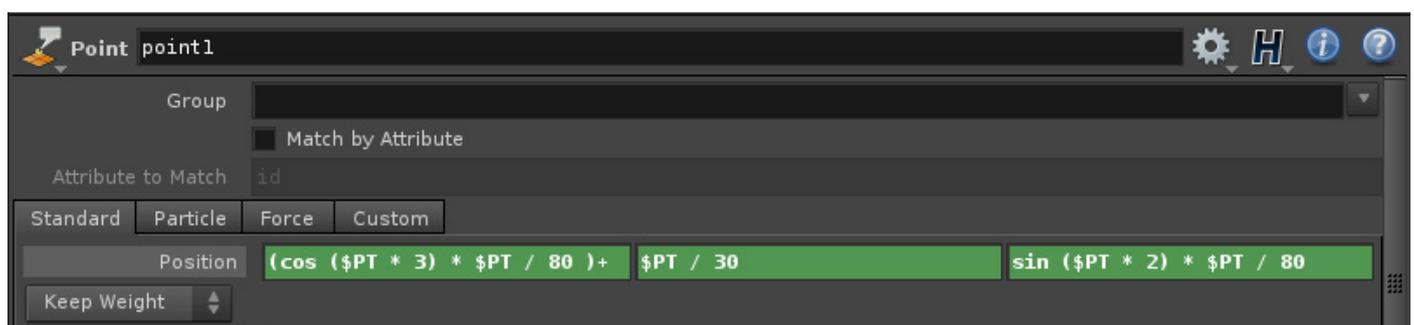
Lorsqu'un numéro de point est appelé et calculé, le résultat s'applique à lui seul. Ainsi chaque point reçoit une valeur différente

* $\$PT / 80$: La curve est composée de 1000 points, numérotés de 0 à 999 de la base à son sommet. En multipliant alternativement l'expression par la valeur des points on obtient une spirale plus étroite en bas et plus évasée en haut.

Nous allons rendre aléatoire le placement des points de l'axe X grâce à la fonction **noise**.

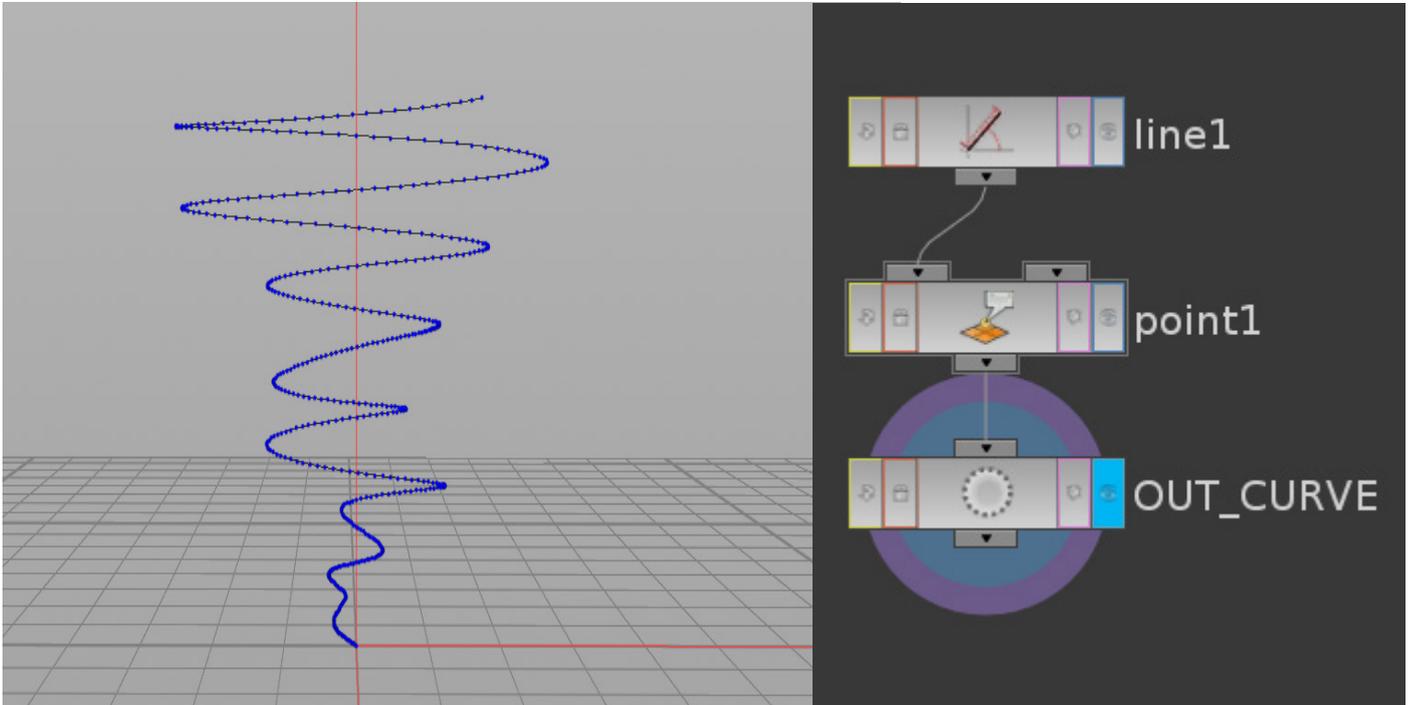
Écrire dans le channel X :

$(\cos(\$PT * 3) * \$PT / 80) + \text{noise}(\$TX, \$TY, \$TZ)$



Enfin placer en dessous du node *point*, un *Null*, le renommer : OUT CURVE.

Les nodes *Null* n'effectuent pas d'opération, ils servent de référence lorsque nous devons importer une succession de nodes dans une autre géométrie. Placé idéalement à la fin de l'arbre de connexions, il sera le premier node interprété par Houdini, car ce dernier li les nodes les plus bas en premier, puis remonte progressivement le long des branches. Si nous modifions les nodes au-dessus du *Null*, l'exportation s'effectuera sans problème, car la référence restera inchangée.



L'ÉMETTEUR :

Les particules seront émises depuis une sphère placée à la base de la spirale.

Dans le shelf *Creat*, faire un **CTRL Clic-Gauche** sur l'icône **Sphère**.

Double cliquer sur le node *sphere_object1* qui s'est créé dans la vue contextuelle **Scene**.

Puis brancher un node *Null* en dessous du node *sphere1*.

Le renommer : OUT EMITTER

ÉMETTRE LES PARTICULES :

L'opération consistant à émettre les particules puis à les diriger le long de la curve, sera effectuée dans une nouvelle géométrie nommée **PARTICLES**. Nous allons donc importer au sein de cette dernière, les objets nommés **EMITTER** et **CURVE**.

Dans la vue contextuelle **Scene**, créer une nouvelle géométrie, la renommer : **PARTICLES**.

Double-cliquer dessus.

Une fois à l'intérieur, supprimer le node *file1*.

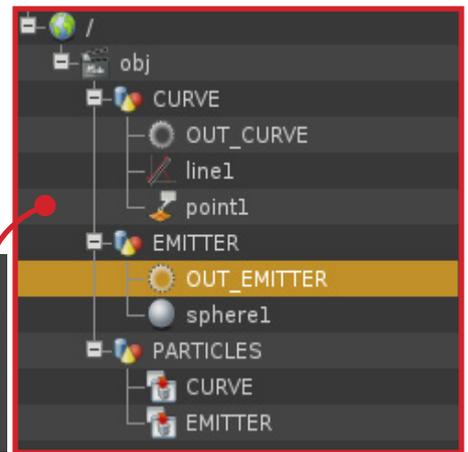
Puis, créer deux nodes *Object merge*.

Renommer le premier : **EMITTER**

le second : **CURVE**

Dans le paramètre *Object1* du node EMITTER, charger le *Null* OUT EMITTER qui se trouve dans la géométrie EMITTER. Vous pouvez effectuer cette opération de deux manières : soit en écrivant dans le champ : `/obj/EMITTER/OUT_EMITTER`.

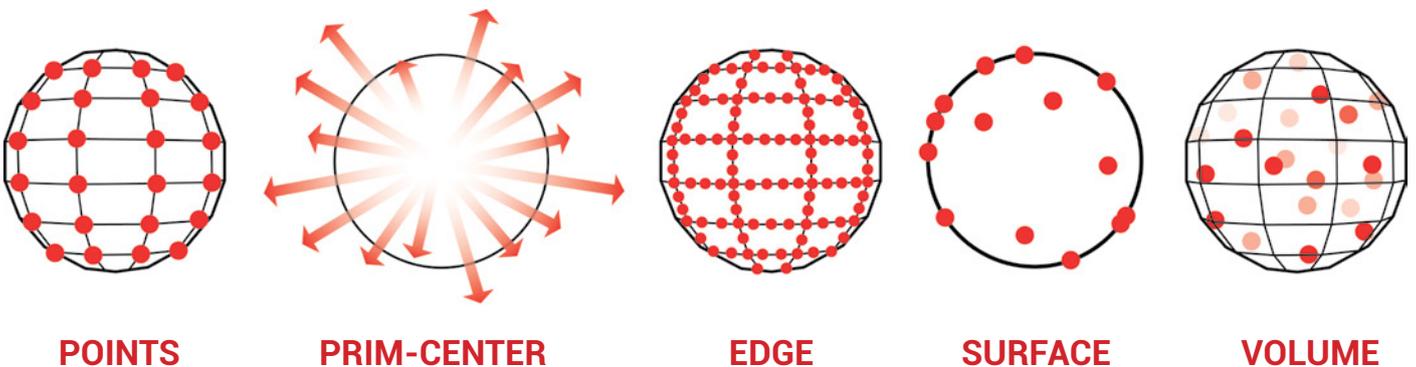
Soit en sélectionnant l'icône  placé à sa droite, puis en allant chercher le *Null* au sein de la hiérarchie. De la même manière, charger dans le paramètre *Object1* du node CURVE, le *Null* OUT CURVE qui se trouve dans la géométrie CURVE.



Cette hiérarchie représente la manière dont les nodes et leurs dépendances s'organisent au sein du fichier.

Paramétrer l'émetteur :

Il y a de multiples façons d'émettre des particules à partir d'une géométrie, voici les principales :



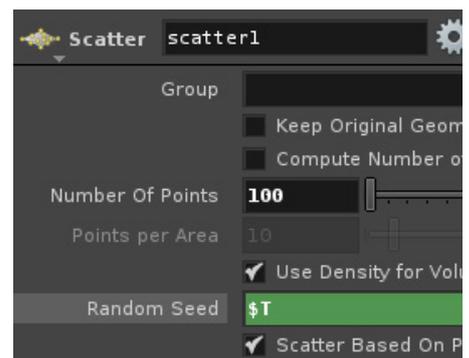
- Point** : émet depuis chaque point de la géométrie.
- Prim Center** : émet depuis le centre de la géométrie.
- Edge** : émet depuis les edges de la géométrie.
- Surface** : émet depuis la surface de la géométrie.
- Volume** : émet depuis un nuage de points créé à l'intérieur de la géométrie.

Pour notre effet, nous utiliserons un mélange entre l'émission par points et par volume. Dans le but d'obtenir un motif d'émission différent à chaque seconde, nous allons générer à l'intérieur de la sphère nommée EMMITER un nuage de point dont nous ferons varier la répartition toutes les 24 images. Chaque point sera alors une source d'émission de particule. Le but étant d'obtenir une distribution hétérogène et donc plus naturelle.

Brancher sous le node EMMITER le premier input d'un node *isoOffset*.

Puis placer l'output de ce dernier dans un node *Scatter*. Régler le paramètre *Number Of Point* du *Scatter* à : **100**. Puis écrire dans son champ *Random Seed* : **\$T**.

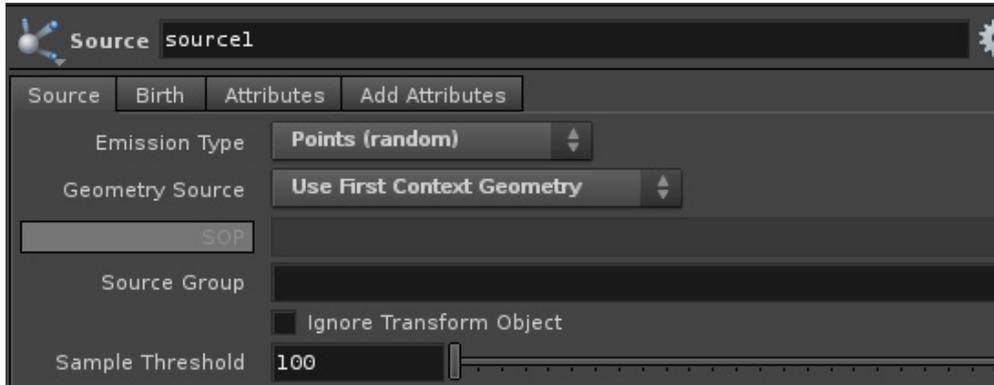
L'*isoOffset* va convertir la sphère en volume, permettant ainsi d'indiquer au scatter où se trouve l'intérieur de l'objet. Le **scatter** répartira alors de manière aléatoire des points dans cet espace. Son paramètre **Number Of Point** permet de contrôler le nombre de points que l'on souhaite créer. Le paramètre **Random Seed** quant à lui, va faire varier la disposition de ces points chaque seconde grâce à la variable **\$T**, qui sera remplacé toute les 24 images par la valeur en seconde de la frame.



Brancher l'output du *Scatter1* dans le premier input d'un node *Pop Network*.
C'est à l'intérieur de ce node que nous allons créer les particules.
Double-cliquer sur ce dernier afin de rentrer dans la vue contextuelle **Particles**.

Afin d'émettre des particules, créer un node *Source*.
Deux de ses onglets nous intéressent : le *Source*, et le *Birth*.
Le *Source* indiquera depuis où seront émises les particules, et le *Birth* indiquera quand.

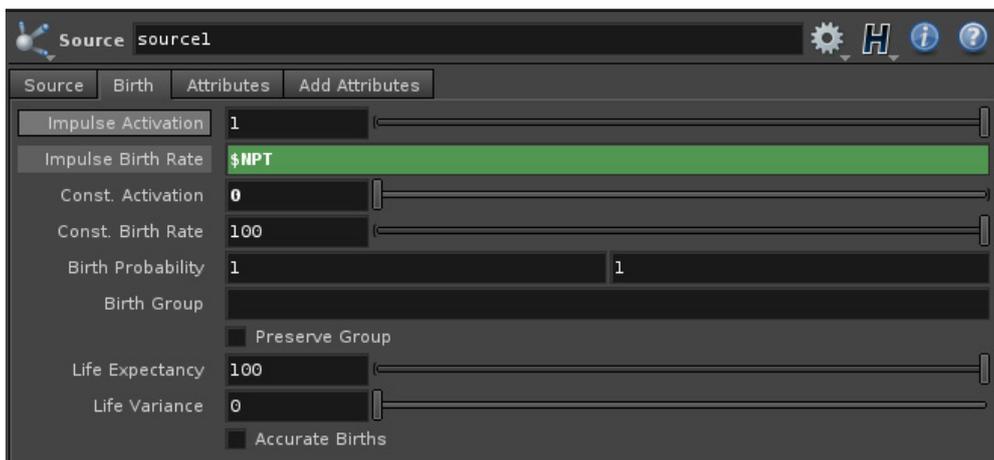
Dans l'onglet *Source*, régler le paramètre *Géométrie Source* sur : **Use First Context Géométrie**.
Puis mettre le paramètre *Emission Type* sur : **Point (random)**.



Le paramètre *Géométrie Source* sert à indiquer au node dans quel input du popnet est branchée la géométrie qui servira d'émetteur. Le réglage **Use First Context Géométrie** pointera alors vers le premier input du node *Popnet1*.

Le paramètre *Emission Type* permet de choisir quel élément de la géométrie émettra des particules (point, edge, surface, etc., se référer au schéma page précédente). Le mode **Points** nous permet de générer des particules à partir des points du scatter.

Dans l'onglet *Birth*, mettre le paramètre *Const Activation* à : **0**.
Puis écrire dans le champ *Impulse Birth Rate* : **\$NPT**.



Il y a deux manières de faire naître des particules :

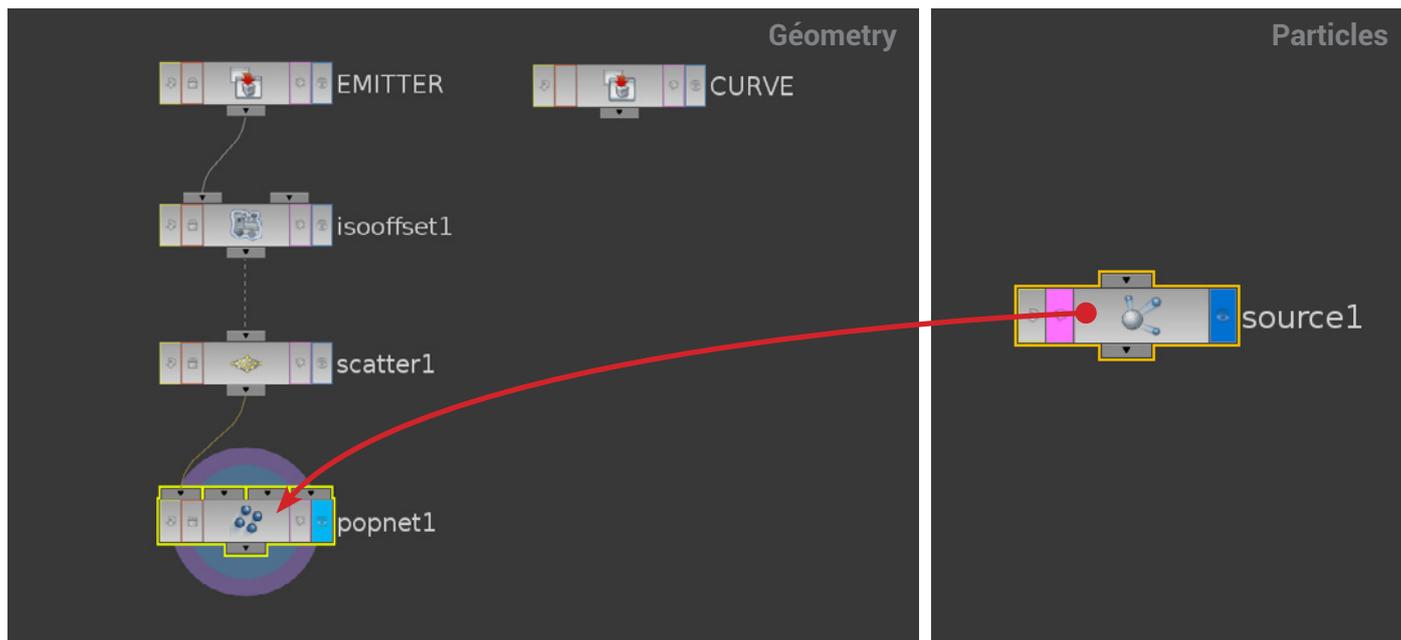
Impulse : Créé un certain nombre de particules à chaque fois que le node est évalué (ça peut être plusieurs fois par seconde)

Constant : Créé un nombre donné de particules à chaque seconde.

Les paramètres *Impulse Activation* et *Const Activation* servent à activer ou non l'un des deux types d'émission. Si la valeur est égale à 0 alors elle sera désactivée, et inversement pour une valeur de 1.

Les paramètres *Impulse Birth Rate* et *Const Birth Rate* servent à régler la quantité de particules à générer.

L'expression $\$NPT$ va renvoyer un chiffre égal au nombre de points présents dans l'émetteur. Ainsi nous allons créer à chaque fois que le node est évalué 100 particules (car il y a 100 points dans le scatter).



FAIRE AVANCER LES PARTICULES LE LONG DE LA CURVE :

Afin que les particules suivent le tracé de la curve, nous allons utiliser ses normales pour leurs indiquer itinéraire.

Dans un premier temps, nous créerons les normales et les dirigerons dans le sens d'inclinaison de la curve. Puis dans un second temps, nous transmettrons les informations de normale aux particules grâce au système du *Point Cloud*.

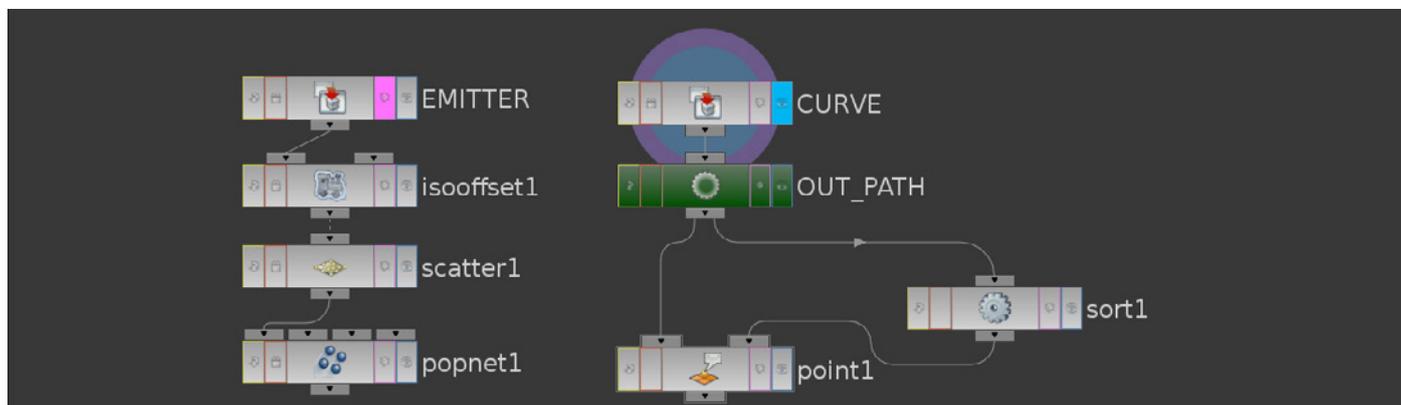
Diriger les normales :

Dans la géométrie PARTICLE, créer un node *Null* sous le node CURVE

Le renommer : OUT_PATH

Relier à son output le premier input d'un node *Point*, ainsi qu'un node *Sort*.

Puis mettre l'output du *Sort* dans le second Input du *Point*.

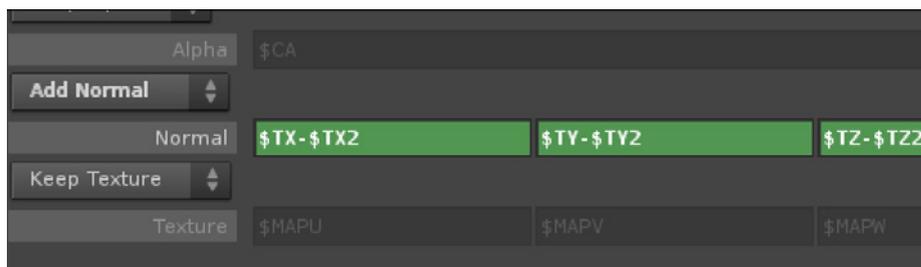


Dans le node *sort1*, régler le paramètre *Point Sort* sur : **Shift**.
Puis mettre une valeur de **1** dans son *Offset*.

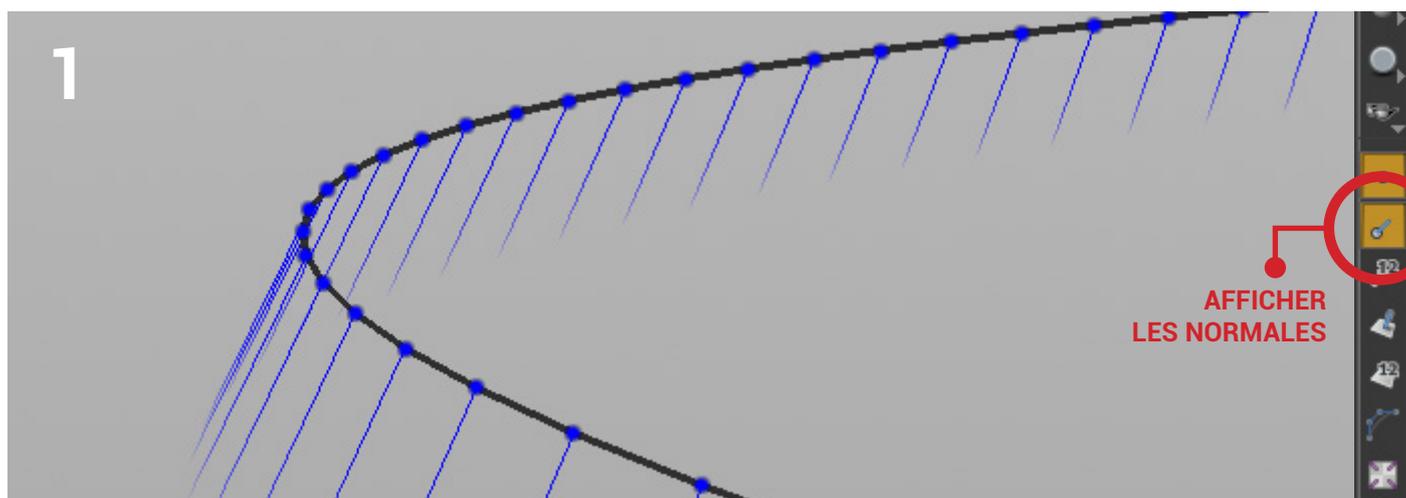
Dans l'onglet *Standard* du node *point1*, mettre le paramètre **Keep Normal** sur **Add Normal**.
Puis écrire à la place de \$NX :
\$TX-\$TX2

À la place de \$NY :
\$TY-\$TY2

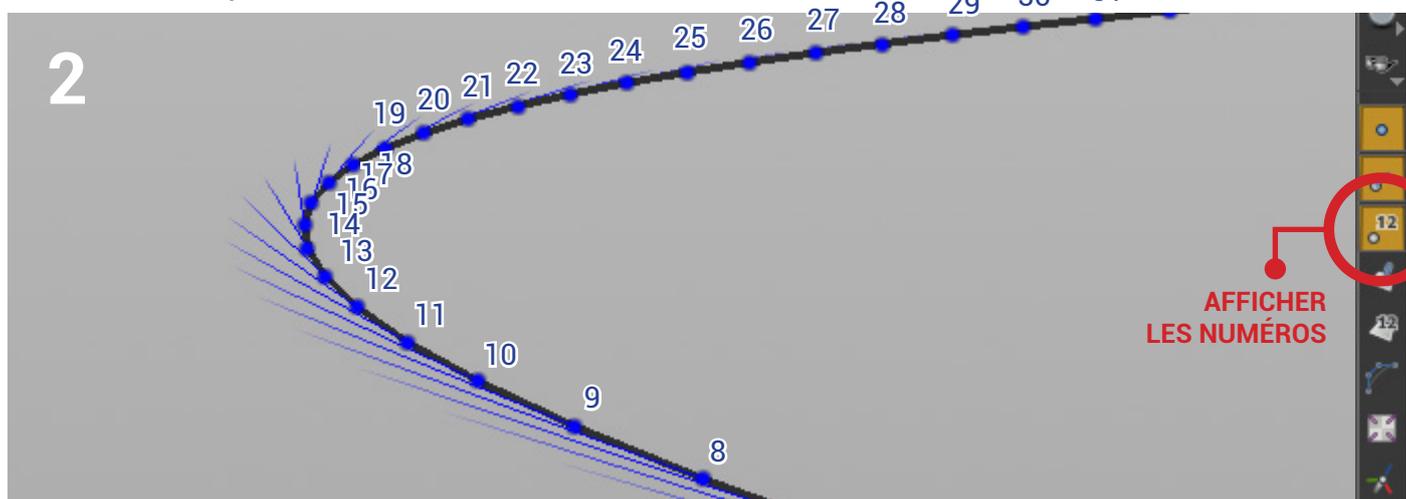
À la place de \$NZ :
\$TZ-\$TZ2



Les normales avant redirection :



Les normales après redirection :



Chaque point de la curve porte un numéro d'identification, le paramètre *Shift* du node *Sort* va décaler la valeur de ce numéro d'autant qu'il est indiqué dans son *Offset*. Le point numéro 8 s'appellera alors numéro 9, et ainsi de suite.

L'intérêt d'une telle opération réside dans son utilisation au sein de l'expression que nous utilisons dans le node *point*.

Pour ne prendre qu'en exemple le channel X et le point 8 : **\$TX - \$TX2**, soustrait la position X du point 8 issu du premier input (TX), à la position X du point 8 du second input (TX2), qui se trouve alors à celle du point 9. Ce qui a pour effet de faire pointer la normale du point vers le point suivant.

Le node va répéter le calcul pour chaque point de la géométrie.

Une fois les normales redirigées, on peut constater qu'il y a une **erreur** sur celle du point 0.

Pour y remédier, nous allons le supprimer.

Brancher sous le node *point1* un node *Delete*.

Mettre son paramètre *Entity* sur **Points**.

Puis écrire dans *Pattern* : **0**

Ainsi le node *Delete* supprimera le point numéro 0.

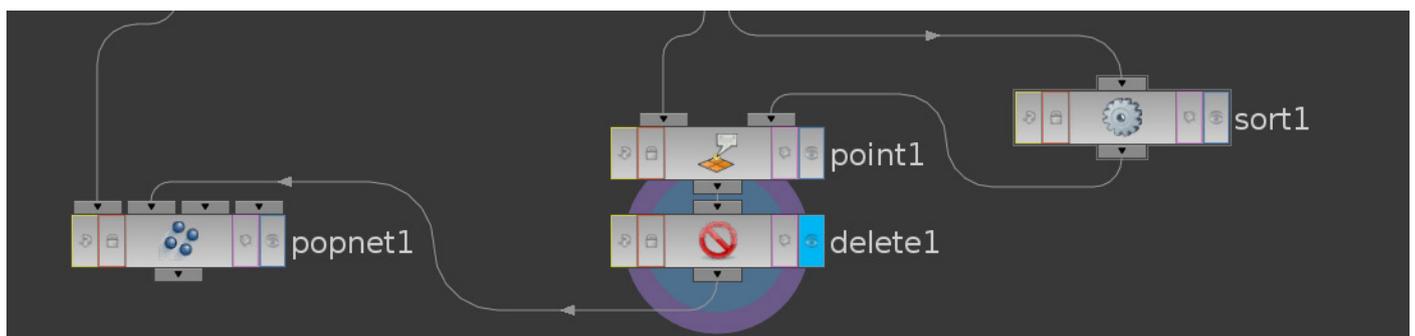
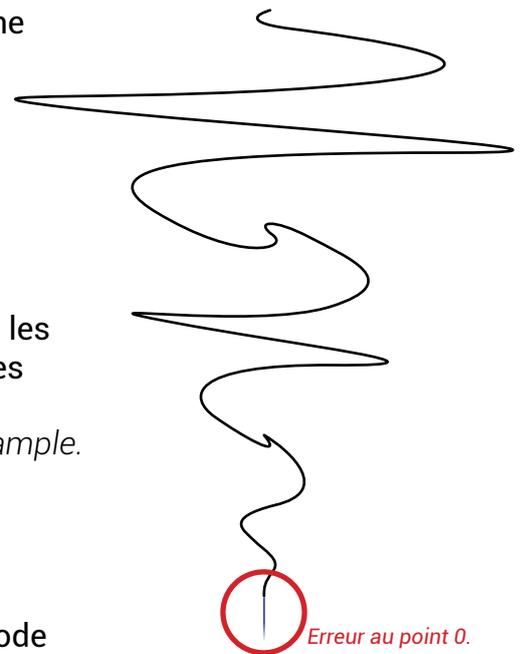
Nous allons à présent ajouter des points sur la curve, afin que les particules aient une plus grande quantité d'informations issues des normales, et puissent se diriger précisément.

Entre le node CURVE et le node OUT PATH, créer un node *Resample*.

Mettre son paramètre *Level of Detail* à une valeur de : **0.03**.

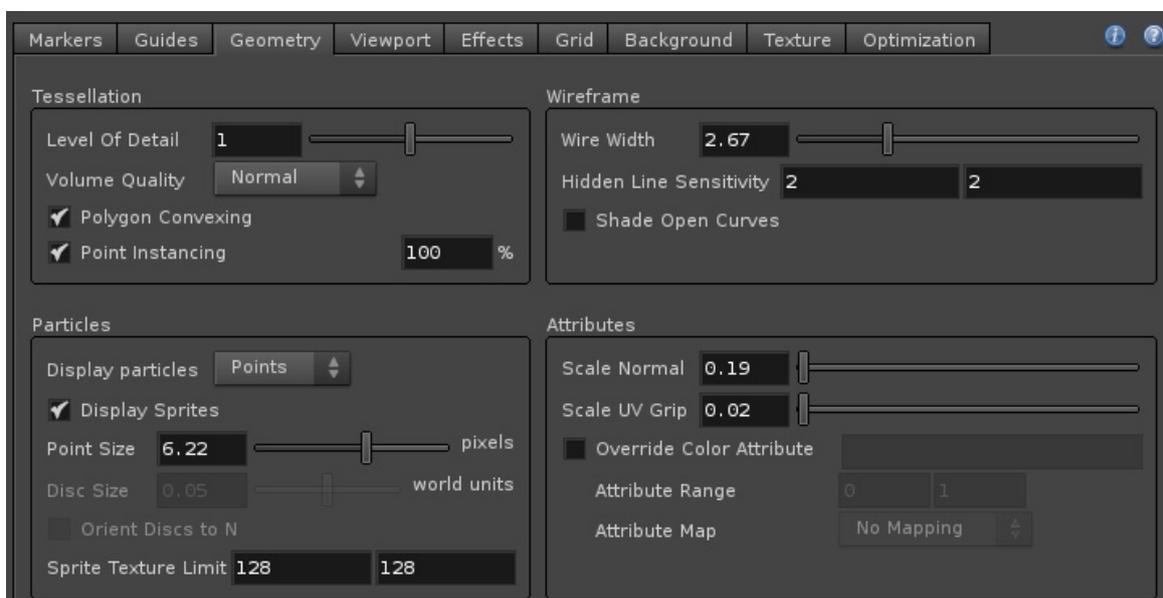
Enfin nous allons intégrer notre curve au *Popnet*, permettant ainsi de l'utiliser dans la vue contextuelle *Particle* :

Brancher le output du node *delete1* dans le second input du node *popnet1*.



ASTUCE :

Placer la souris dans la *Scène View*, et presser la **touche D** du clavier afin d'afficher la fenêtre *Display Option*. L'onglet *Géométrie* vous permettra alors de modifier l'affichage des points, normales et lignes.



Créer la simulation :

Rentrer dans le node *Popnet*, puis brancher en dessous du node *Source1* un node *VOP POP*. C'est dans ce dernier que nous allons mettre en place le *Point cloud* nous permettant de diriger les particules le long de la curve.

Dans notre cas, le *Point cloud* peut-être comparé au node *Attribut transfer*, qui transfère les attributs d'une géométrie sur une autre dans un périmètre défini par l'utilisateur.

Rentrer dans le *VOP POP*.

Créer un node *Point Cloud Open*.

Au sein de ce dernier, nous allons utiliser 3 paramètres : le *Point Cloud Texture*, le *Search Radius* et le *Number of Points*.

Le paramètre **Point Cloud Texture** indique depuis quelle géométrie transférer les attributs.

Le paramètre **Search Radius** correspond au périmètre dans lequel cette géométrie peut transférer ses attributs.

Et enfin le paramètre **Number Of Point** définit le nombre maximum de points que le node va retourner.

Pour plus de confort, nous allons envoyer ces paramètres à la surface du node, afin de pouvoir les retoucher sans avoir à rentrer dans le *VOP POP*.

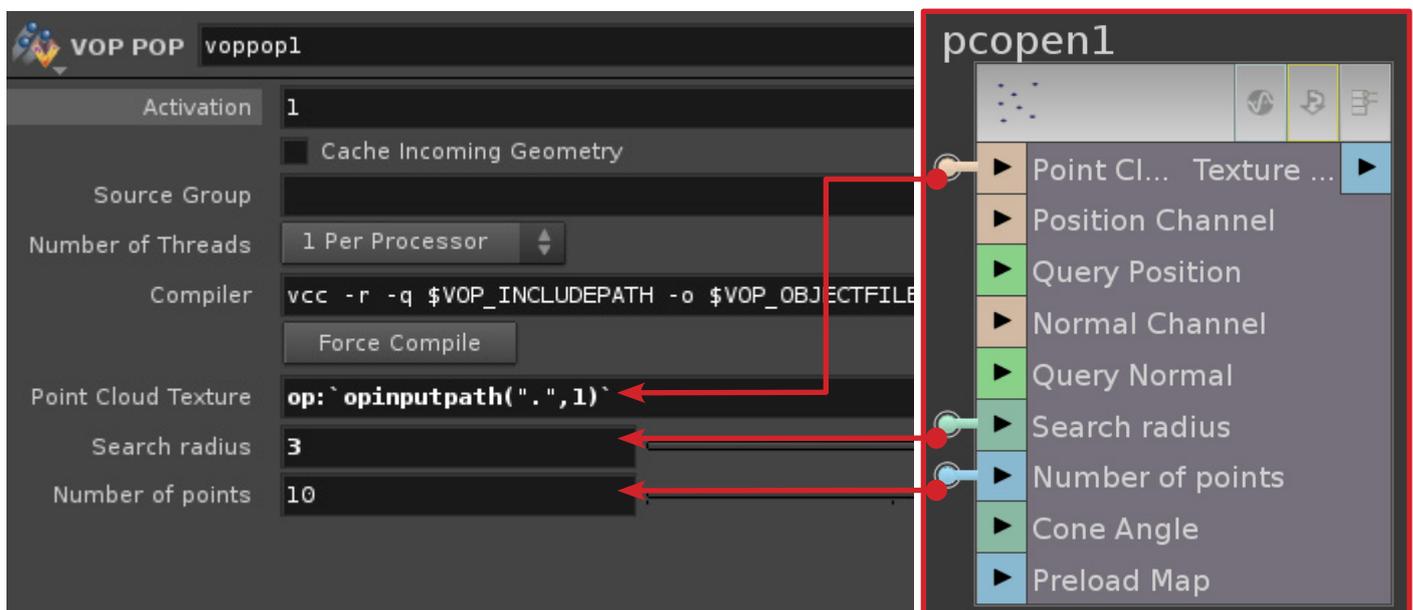
Pour ce faire, effectuer un **Clic-du-milieu** sur le premier input du node *Pcopen1*, puis sélectionner : **Promote Parameter**. Faire de même pour le *Search Radius* et le *Number Of Points*.

Puis sortir du node afin de les paramétrer.

Écrire dans le champ du *Point Cloud Texture* : `op: `opinputpath(".",1)``

dans le *Search Radius* : **3**

dans le *Number Of Points* : **10**



`op` : `op` va permettre à Houdini de lire un node à la place d'un fichier. Car le point cloud texture est fait pour lire un fichier `.pc`, or nous souhaitons lui faire utiliser un nuage de point qui se trouve encore dans Houdini, et qui n'a donc pas été exporté en `.pc`.

`opinputpath()` : permet de sélectionner dans la hiérarchie un node connecté à un input spécifique, ici nous souhaitons utiliser la géométrie qui se trouve dans le second input du node *vopsop1*. Cette expression a besoin de deux paramètres pour savoir où chercher : le nom/chemin

du node et le numéro de l'input dans lequel est connecté la géométrie recherchée. C'est ce qu'indiquent les éléments entre parenthèses : ("..",1).

".." : Signifie dans le node qui me contient. C'est-à-dire le *popnet1*.

1 : Indique dans quel input du *popnet1* lire les informations, il représente le second input (le premier étant nommé 0).

Les Backticks : `` Que l'on retrouve avant et après le *opinputpath* sont à ne pas confondre avec des guillemets. Ils permettent d'exécuter une expression au sein d'une chaîne de caractères.

L'expression va donc chercher la géométrie qui se trouve dans le second input du node *popnet1*.

Retourner à l'intérieur du *Voppop1*.

Relier à l'output du *pcopen1* un node *Point Cloud Filter*.

Écrire dans son paramètre *Channel* : **N**

Ce node va permettre de sélectionner l'attribut **N (normale)** issu du *Point Cloud Open*.

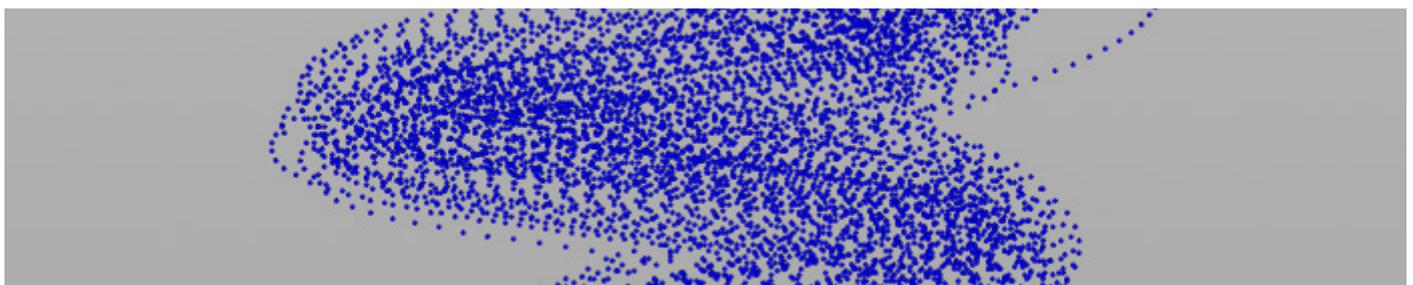
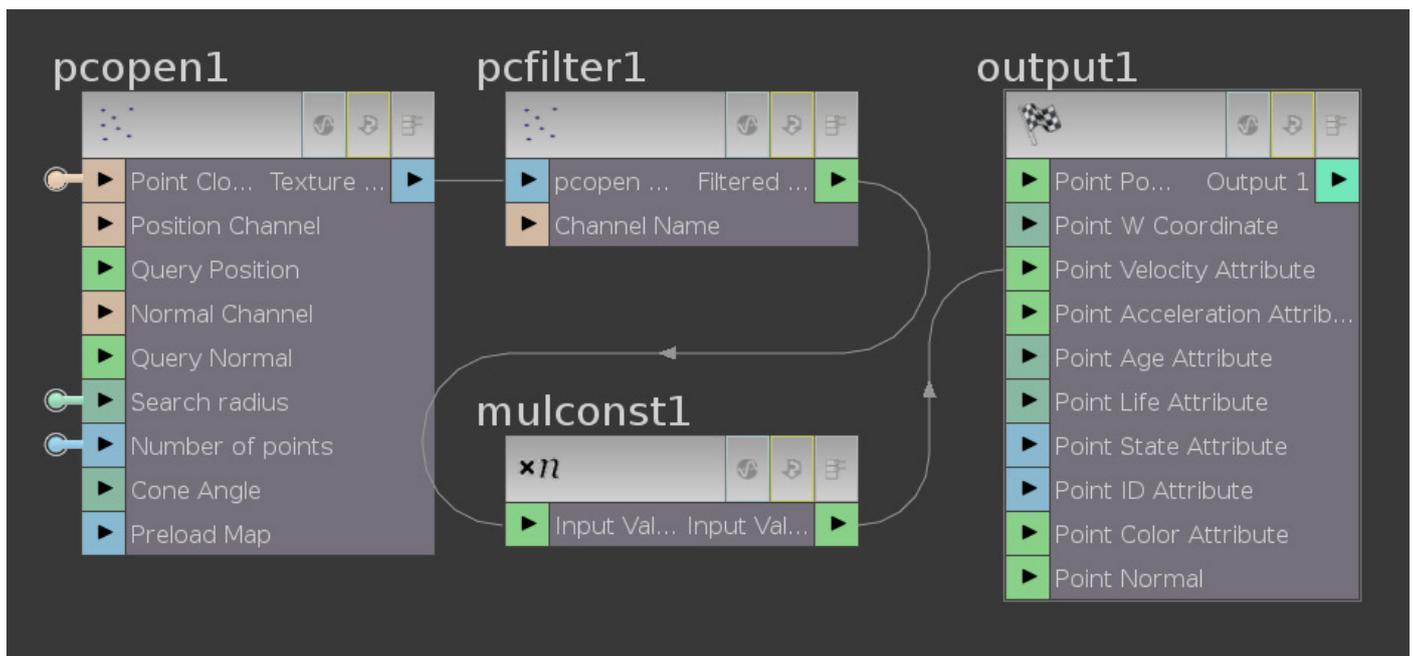
Enfin relier sa sortie à l'input *Point Velocity* du node *output1*.

En théorie si on lit l'animation, les particules devraient suivre la curve, or il n'en est rien.

Pour y remédier, il suffit de multiplier la valeur du *point cloud filter*. Placer donc un node *Multiply Constant* entre le *pcfilter1* et le node *output1*, et mettre la valeur du *Multiplieur* à **100**.

Puis, mettre une valeur de **100** à son paramètre *Multiplieur*.

Les particules suivront à présent la curve.



Ajouter du bruit dans la position des particules :

Pour le moment le résultat n'est pas naturel, les particules se déplacent en effet avec trop de régularité, ce qui forme une trame peu esthétique. Nous allons donc ajouter des mouvements aléatoires aux particules.

Dans le *VOP POP*, ajouter un node *Curl Noise*.

Brancher la sortie nommée *Point Position* du node *global1* à l'entrée *Position* du *CurlNoise1*.

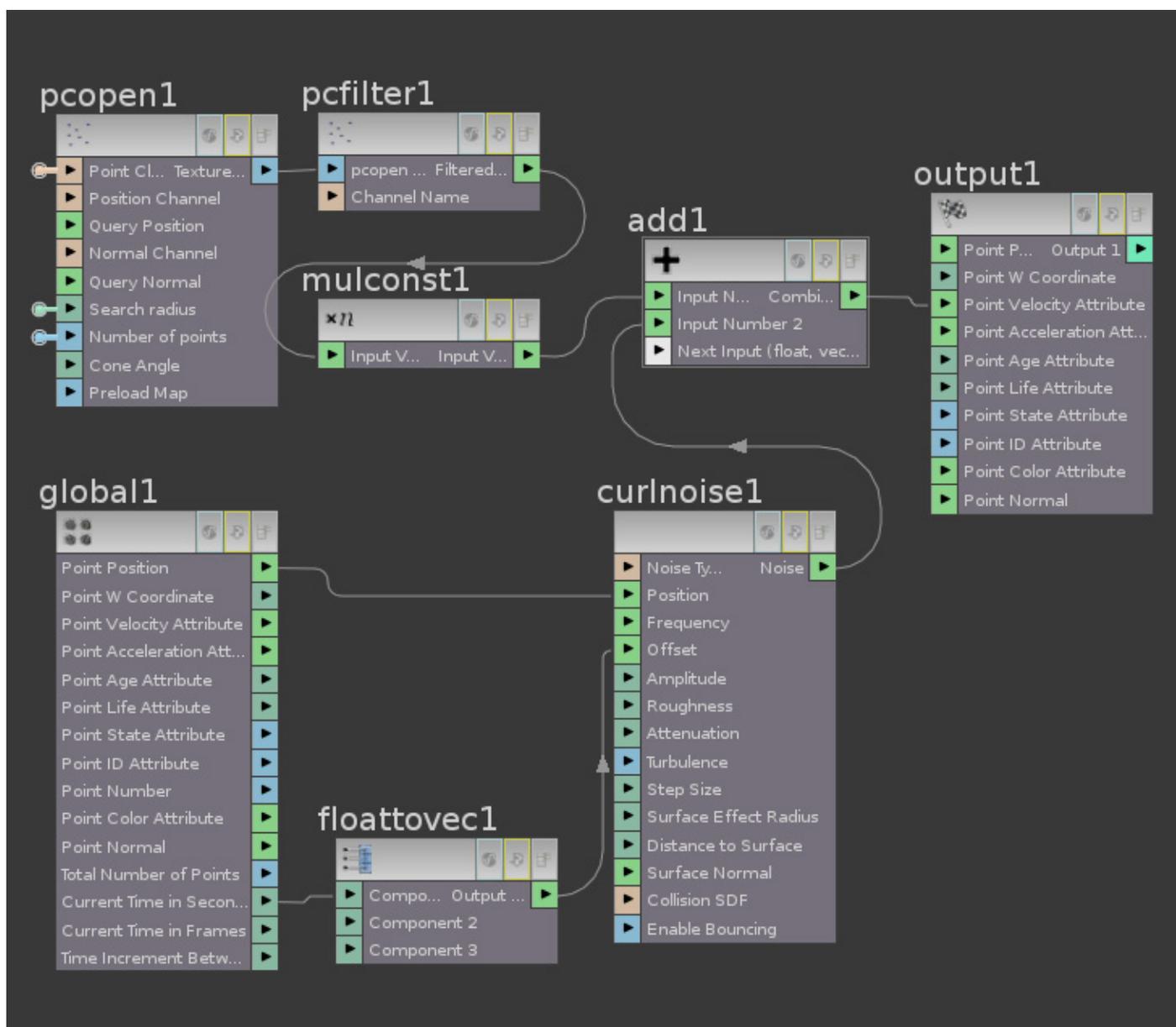
Puis connecter le *Current Time* du *global1* à l'*Offset* du *curlnoise1*.

Nous pouvons constater que la ligne qui les relie est en pointillés. Ce qui indique que la valeur de l'un n'est pas compatible avec celle de l'autre. En effet, l'un est un float, tandis que l'autre est un vector.

Pour les rendre compatibles, il suffit d'ajouter entre eux, un node *float to vector*, qui convertira l'attribut *Current time* en vector.

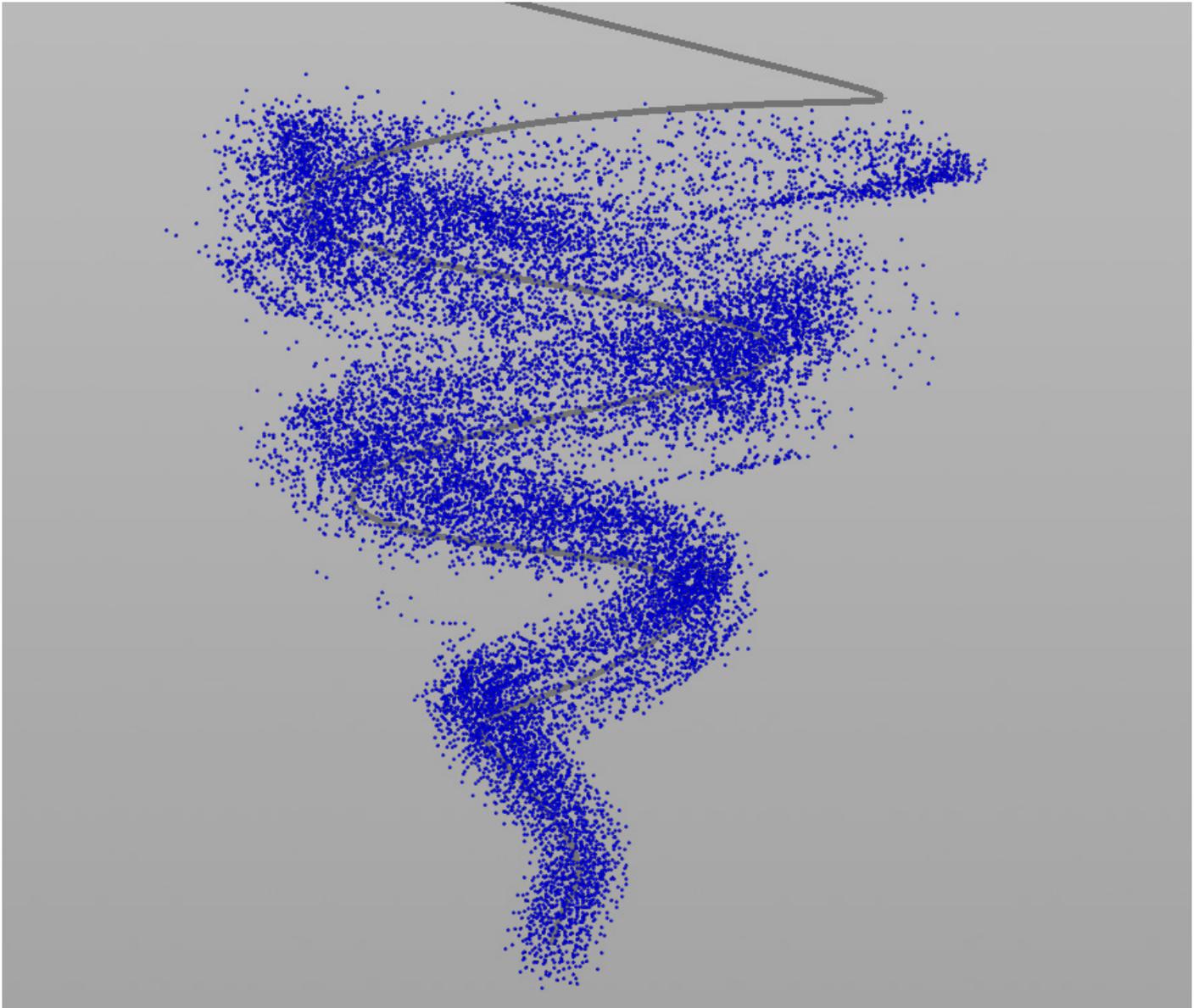
Afin d'ajouter le *Curl Noise* à la vitesse actuelle des particules, placer un node *Add*, entre le *multconst1* et le *output1*.

Puis relier la sortie du *curlnoise1* à la seconde entrée du node *add1*.



Rapprocher les particules de la curve :

Les particules ont tendance à s'écarter de la curve au fur et à mesure qu'elles avancent.



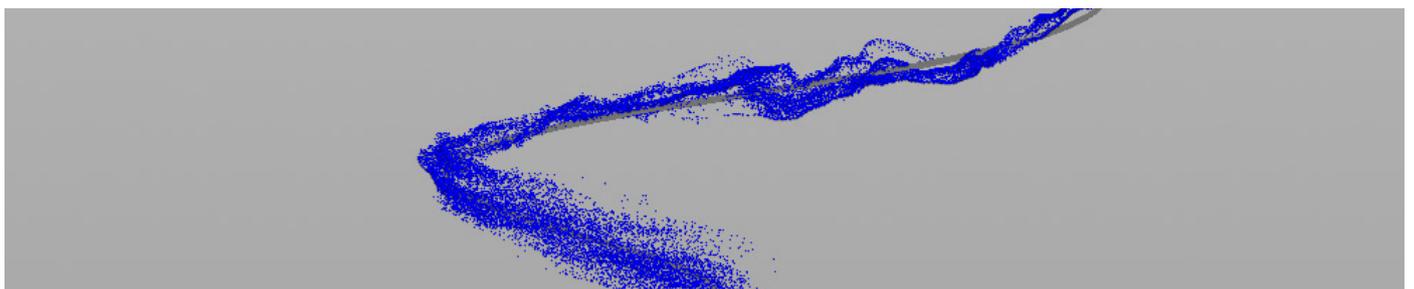
Pour qu'elles s'en approchent, nous allons soustraire la position des points de la curve à celle des particules.

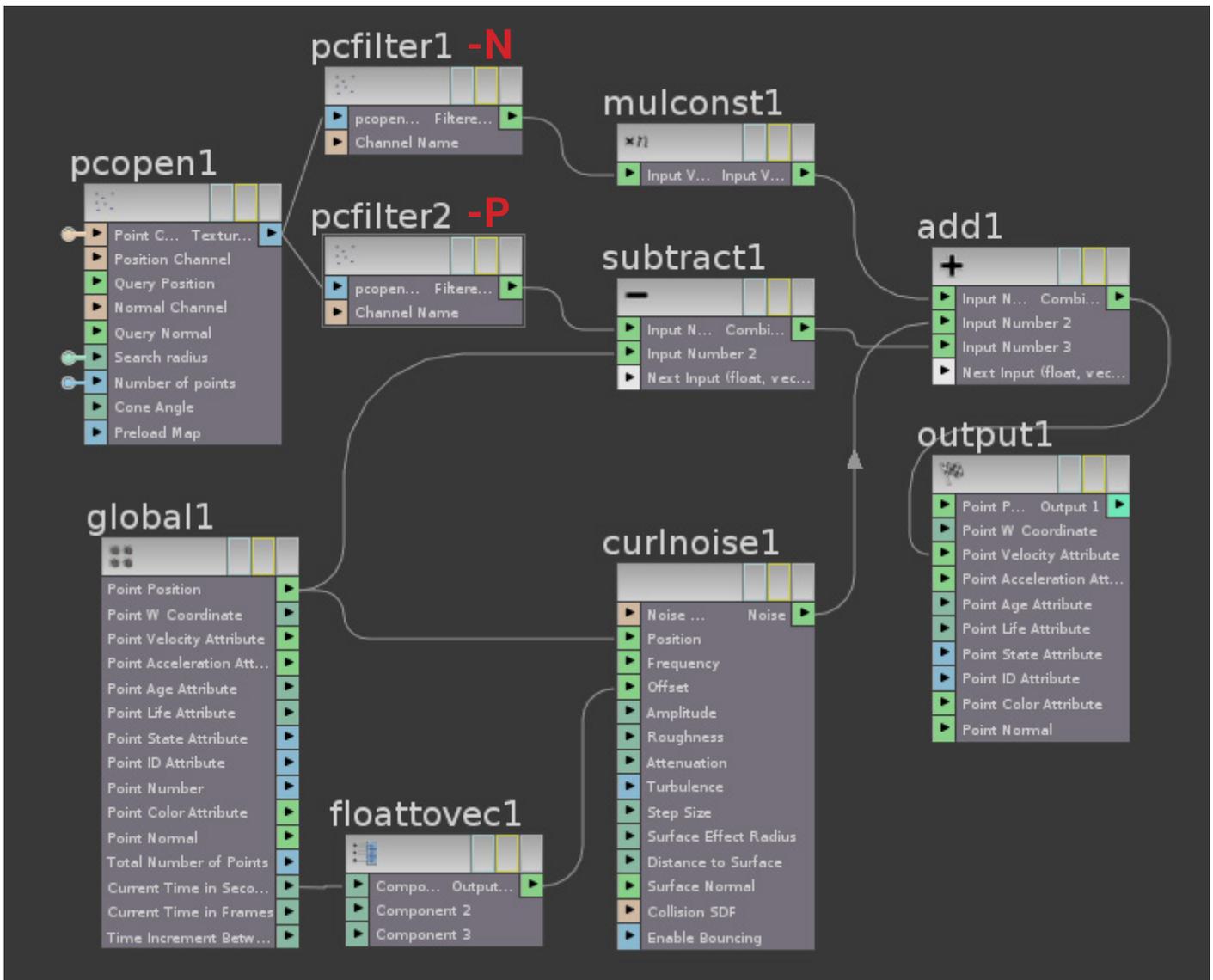
Afin de récupérer la position des points de la curve, créer un node *Point Cloud Filter*, laisser son paramètre *Channel* sur **P** (position).

Puis brancher son output à un node *Subtract*.

Placer dans le second input de ce dernier la sortie *Point Position* du node *global1*.

Enfin, relier le output du node *subtract1* au *Next Input* du node *add1*.





Faire tourner les particules autour de la curve :

Pour cette ultime opération au sein du *VOP POP* nous allons devoir créer un nouvel attribut qui mesure la direction de chaque edge composant la curve.

Retourner dans la géométrie *PARTICLE*.

Placer entre les nodes *delete1* et *popnet1* un node *Point*.

Dans l'onglet *Force*, mettre le paramètre *Keep Edge Force* sur *Add Edge Force*.

Revenir dans le *VOP POP*, créer de nouveau un node *Point Cloud Filter*.

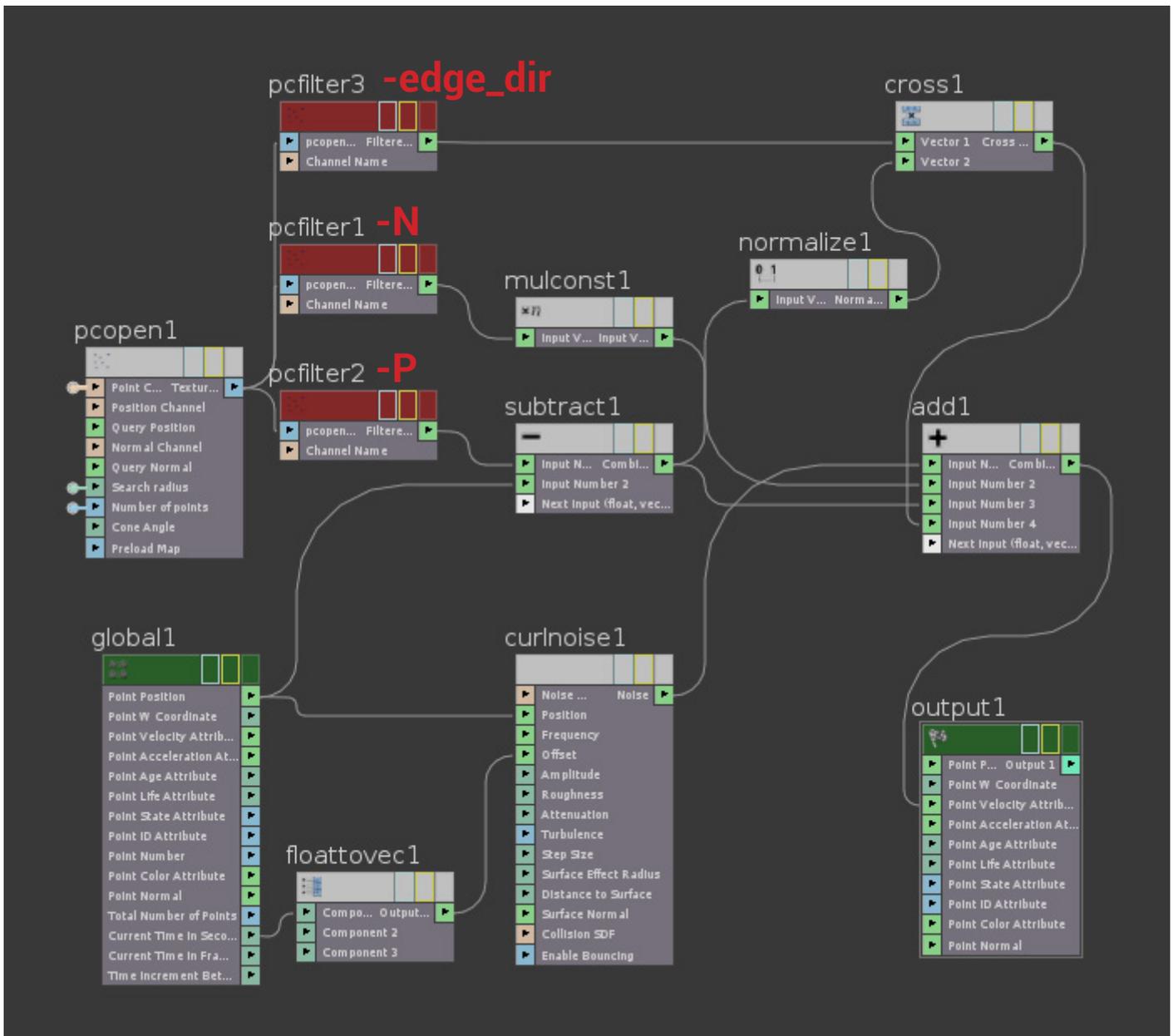
Écrire dans son paramètre *Channel* : **edge_dir**

Cet attribut est celui créé par le node *Point*.

Créer un node *Cross Product*, puis brancher dans son input nommé *Vector1* le node *pcfilter3*, enfin mettre la sortie du node *subtract1* dans son second input.

Ajouter un node *Normalize* entre le *subtract1* et le *cross1*, et brancher la sortie du node *cross* au dernier input du *Add1*.

Le *Cross Product* va effectuer un produit en croix entre l'attribut **edge_dir** de chaque point de la curve, et le résultat issu du node *subtract* qui correspond à la distance entre une particule et la curve. Le node *Normalize* va remapper les valeurs issues du node *subtract* à des valeurs comprises entre 0 et 1.



Vers plus de variation, ajouter du mouvement à la curve :

Retourner dans la géométrie PARTICLE.

Placer entre les nodes *resample1* et *OUT CURVE* un node *mountain*.

Il va déplacer les points de la curve le long de leurs normales par le biais d'un fractal noise.

Mettre son paramètre *Fractal Depth* à 1

son *Roughness* à 0.7

Écrire dans chaque channel du paramètre *Frequency* : 0.3

du paramètre *Offset* : \$T

Et enfin, régler le *Noise Type* sur : **Sparse Convolution**.

Le paramètre *Offset* sert à animer le *mountain*, l'expression \$T va appliquer une valeur qui équivaut à celle de la frame courante.

FRACTURE : PRÉFRACTURER & SIMULER (H 12)

GLUE CONSTRAINT

Dans ce tutoriel, nous allons créer un sol qui s'affaisse.

Avant toute chose, nous préfracturons la géométrie. Afin que ces fractures ne se détachent pas les unes des autres dès la première frame, nous les lierons par des contraintes de glue. Puis nous créerons un système capable de supprimer ces contraintes afin d'enclencher la chute des différents blocs composant la géométrie.

Fichier joint : GlueConstraint.hip

PRÉFRACTURER :

Créer le sol :

Dans le shelf **Creat** faire un **Ctrl + Clic-Gauche** sur l'option *Box*.



Dans le *Network Editor* double-cliquer sur le node *box_object1*.

Régler le **Size** du node *box1* comme suit : **19 0.6 26**

Renommer le node : **Ground**

Mettre en place les fractures :

Afin de fracturer une géométrie, Houdini utilise l'algorithme de Voronoï. Il va générer un ensemble de cellules (voir page 71) sur une surface, en se basant sur la position de points répartis dans l'espace. L'algorithme de Voronoï se manifeste dans Houdini sur la forme du node *Voronoi Fracture*, il a besoin pour fonctionner d'une géométrie et d'un nuage de point (créé grâce au node *scatter*.)

Le scatter :

Toujours à l'intérieur du node *box_object1*, créer une sphère.

Mettre son *Radius* à : **15 15 15**

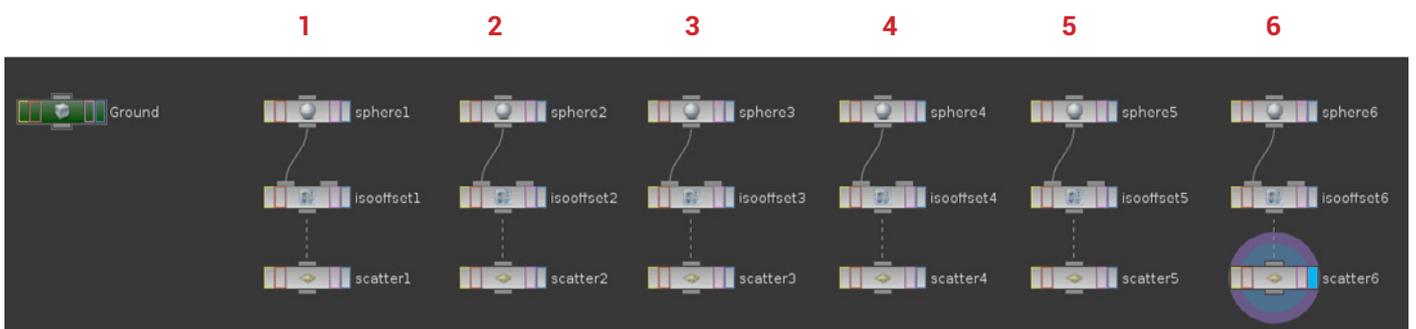
Brancher l'output de la sphère dans le premier Input d'un node *IsoOffset*.

Placer en dessous de l'*IsoOffset* un node *scatter*.

Régler son paramètre *Number Of Points* à **150**.

Enfin sélectionner ces trois nodes, faire **CTRL + C**.

Puis, cinq fois **CTRL + V**.



Nous allons paramétrer chacun de ces ensembles de node :

2 :

Sphère 2 :

Radius : **8 8 8**

Center : **0 0 0**

Scatter 2 :

Number Of Points : **200**

3 :

Sphère 3 :

Radius : **2 2 2**

Center : **0 0 -0.2**

Scatter 3 :

Number Of Points : **300**

4 :

Sphère 4 :

Radius : **2 2 3**

Center : **0 0 0**

Scatter 4 :

Number Of Points : **600**

5 :

Sphère 5 :

Radius : **13 13 18**

Center : **0 0 0**

Scatter 5 :

Number Of Points : **400**

6 :

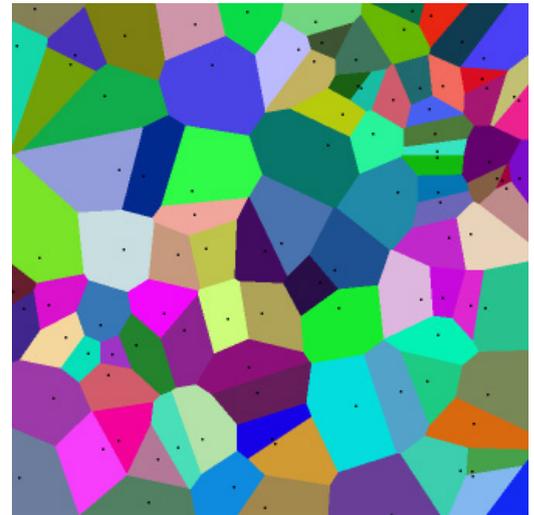
Sphère 6 :

Radius : **5 5 5**

Center : **0 -0.3 0**

Scatter 6 :

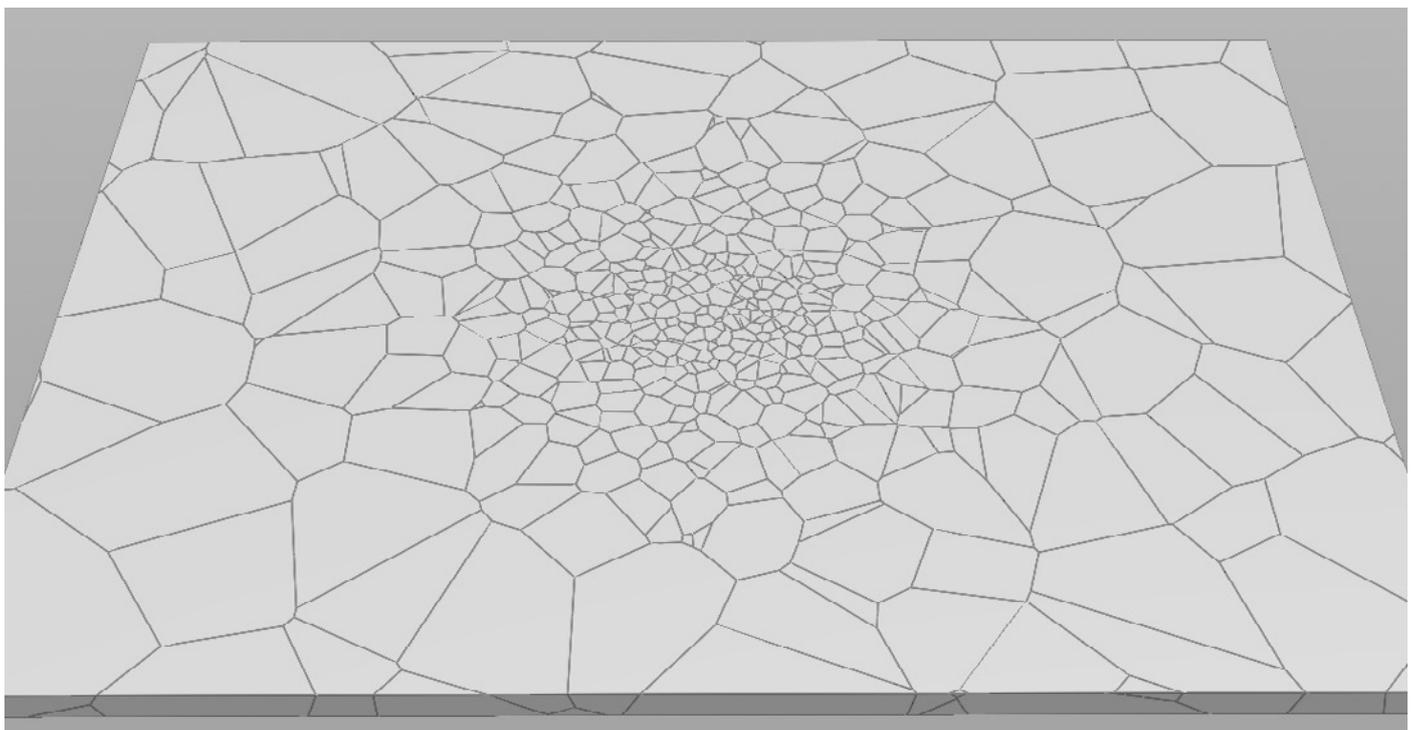
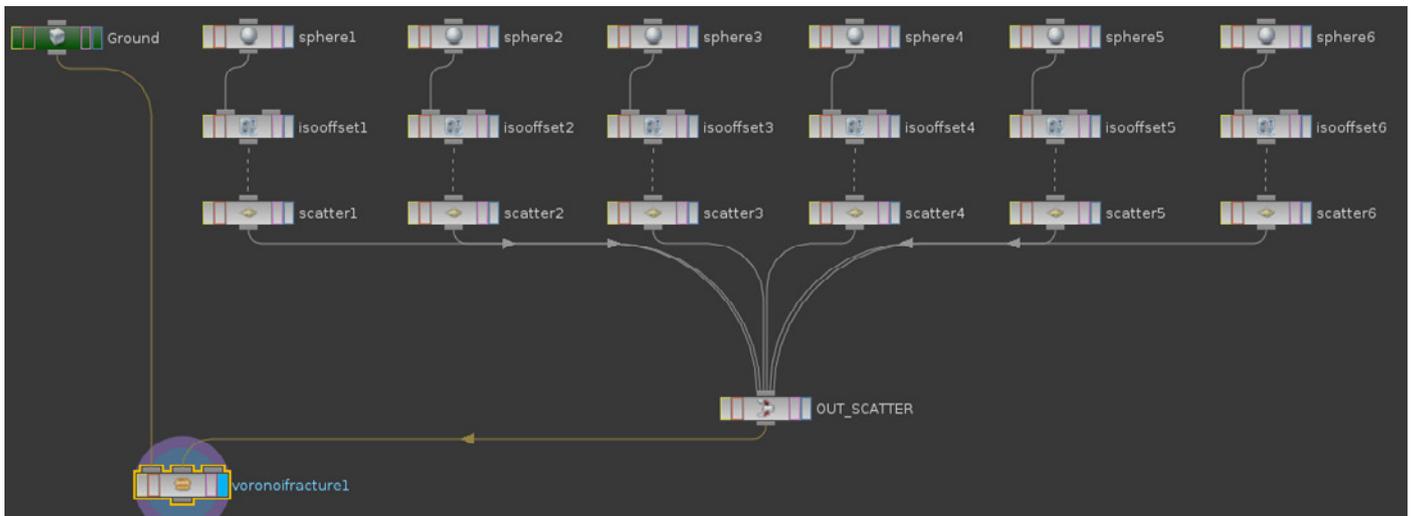
Number Of Points : **1300**



Exemple de cellules de Voronoï, on peut distinguer en noir les points ayant servi à les engendrer.

Relier chacun des outputs des *Scatters* à un node *Merge*. Le renommer **OUT SCATTER**.

Créer un node *Voronoi Fracture*, relier à son premier input le node *Ground*, et à son second input le node **OUT SCATTER**.



Note :
Le node *isoOffset* permet de disposer des points à l'intérieur de la sphère. Il va convertir la géométrie en volume, afin que le scatter puisse identifier l'intérieur de l'extérieur de l'objet.

Sélectionner la zone simulée :

Nous souhaitons uniquement faire chuter les pièces situées au centre du Ground. Nous allons donc les sélectionner, puis les séparer du reste de la géométrie afin de leur appliquer une gravité. La sélection se fait indirectement : dans un premier temps nous créerons une sphère au centre du nuage de points, les points contenus à l'intérieur d'elle seront ensuite ajoutés à un groupe. Dans un second temps, chaque débris généré par ces points recevra un attribut nommé *active*. Puis un node delete supprimera toute géométrie ne possédant pas cet attribut, isolant ainsi les pièces sélectionnées du reste du mesh.

À l'intérieur du node *box_object1*, créer une sphère.

Mettre son *Radius* à : **5 5 5**

Créer un node *Group Géométrie*, le placer entre les nodes *Vornoifracture1* et *OUT_SCATTER*.

Brancher dans son second input le node *sphere7* nouvellement créé.

Renommer le groupe : **Group point select**.

Dans ses paramètres, écrire au champ *Group Name* : **selected**.

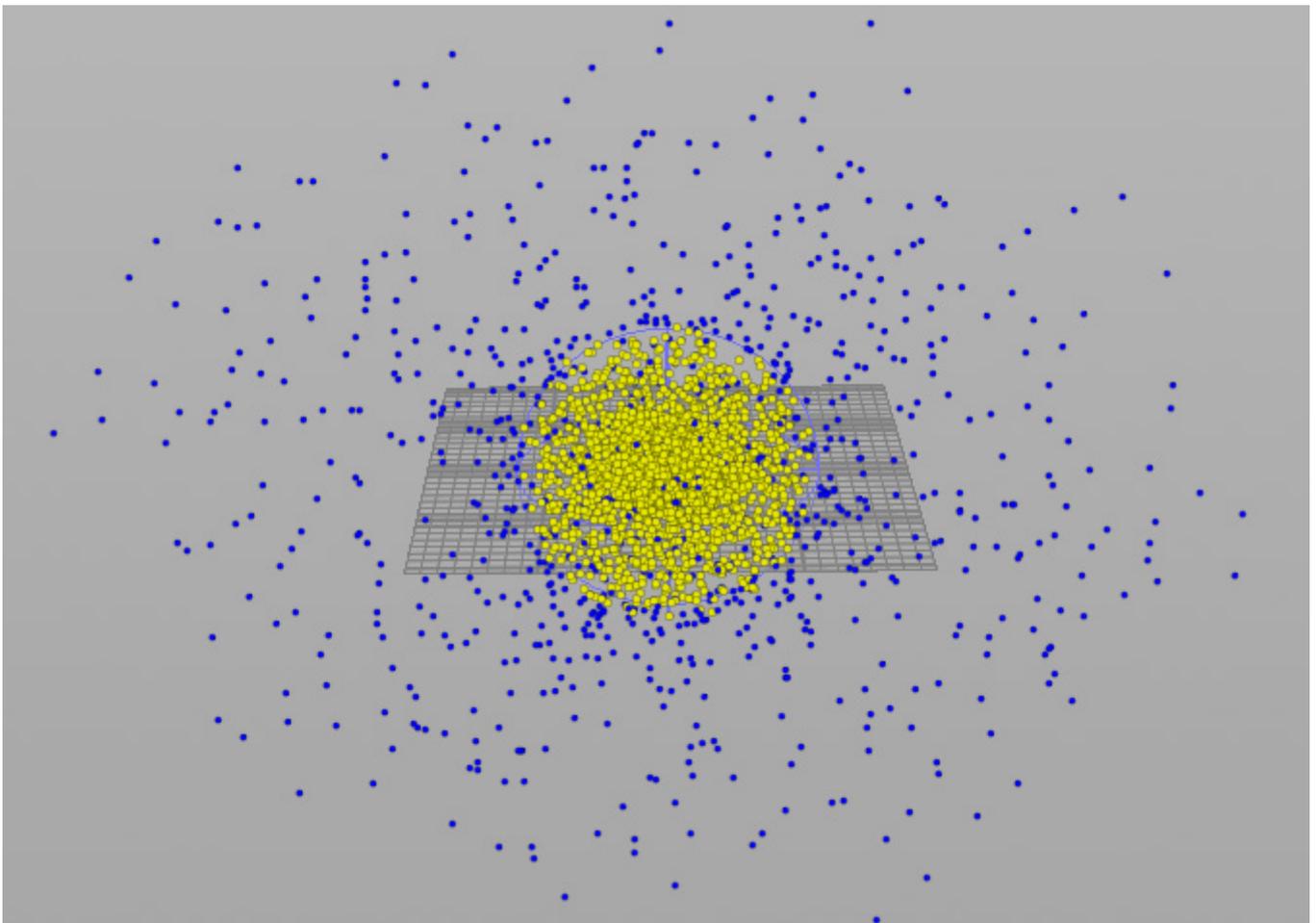
régler le *Entity* sur : **Point**.

Dans l'onglet **Number**, désactiver le paramètre *Enable*.

Dans l'onglet **Bounding**, l'activer.

Mettre le paramètre *Bounding Type* sur : **Bounding Object (point only)**.

Cette manipulation a pour effet de créer un groupe nommé *selected* composé des points situés à l'intérieur de la *sphere7*.



Les points en surbrillance au centre du nuage de point sont ceux sélectionnés par la *sphere7*, ils font partie du groupe *Selected*. Chaque débris généré par ces points tombera lors de la simulation.

Placer entre les nodes:

Group_point_select et *voronoifrac-*
ture1, un node *Attribut creat*.

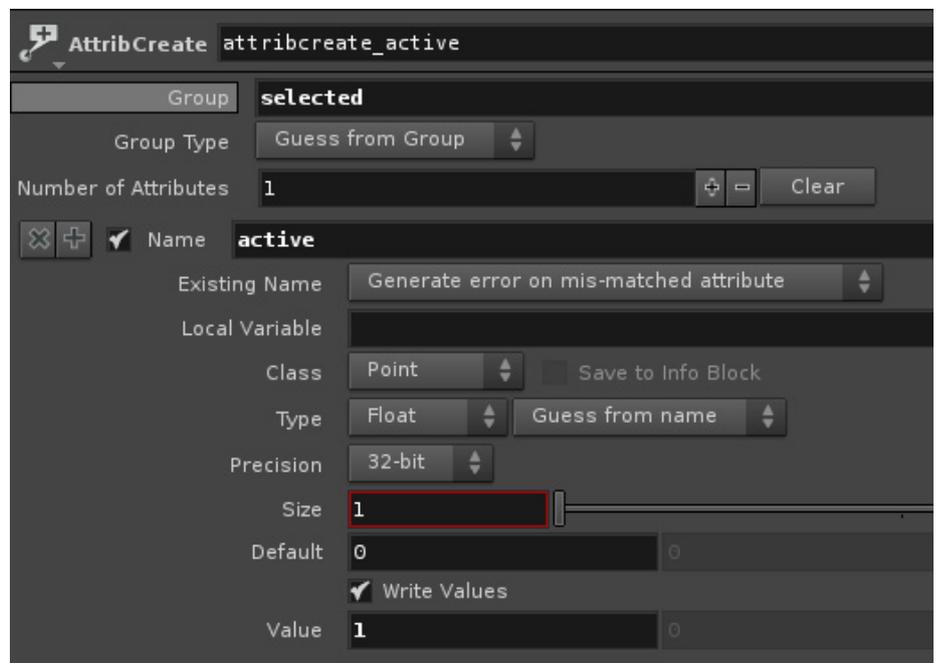
Le renommer **attribcreat_active**.

Dans ses paramètres, charger
à *Group*, le groupe **selected**.

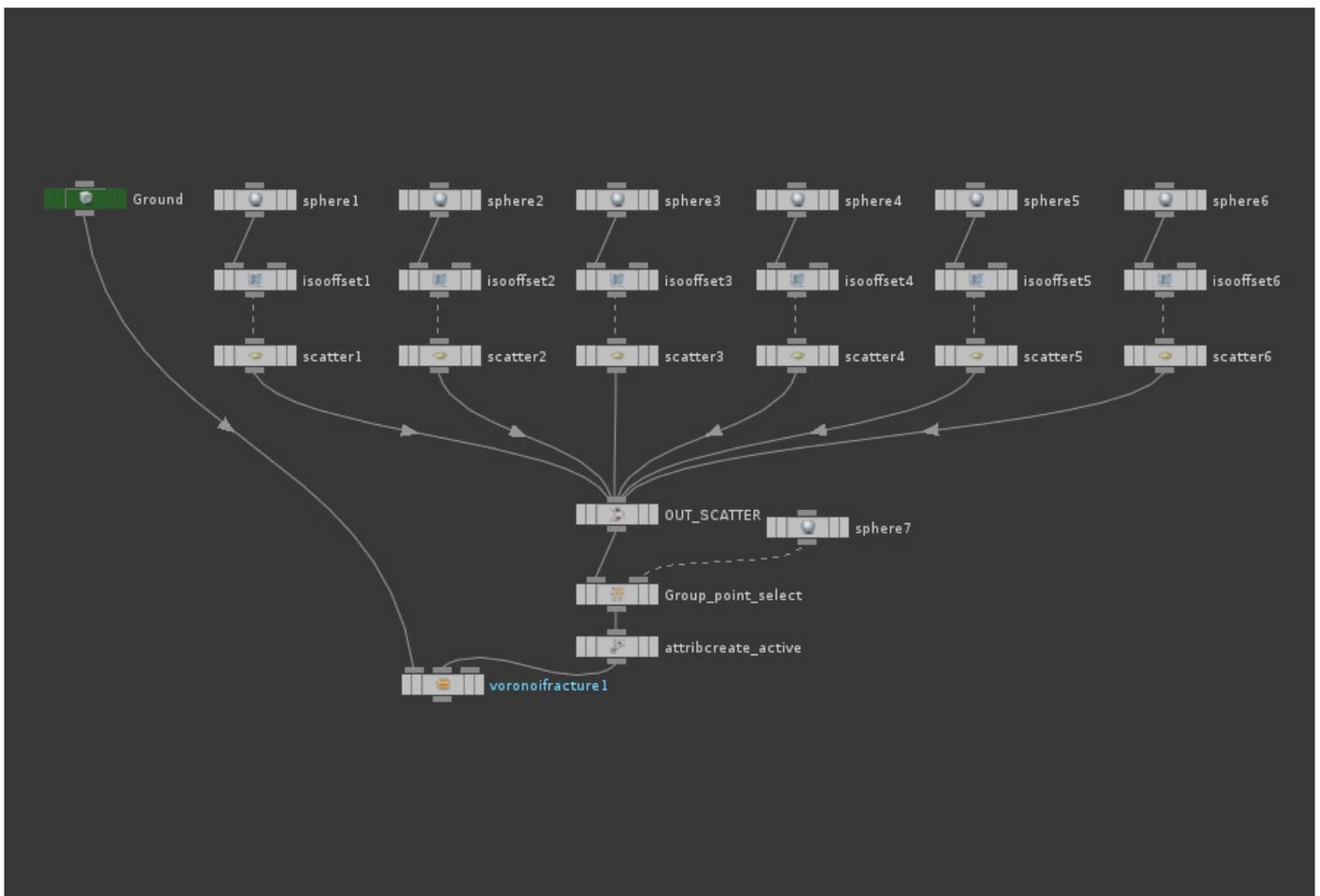
Écrire à la place de *attribut1*
dans le paramètre *Name* :

active.

Enfin mettre dans *Value* : **1**.

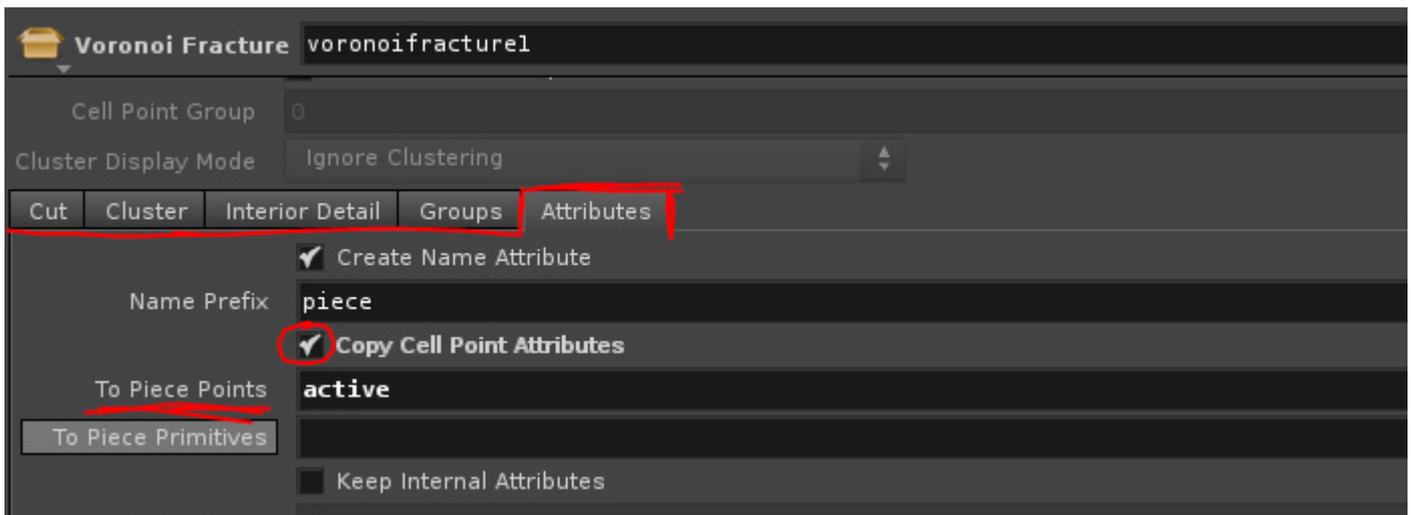


Ce node va affecter aux points compris à l'intérieur du groupe *selected*, un attribut de type point appelé **active**. On le réutilisera plus tard pour supprimer toutes les pièces qui n'en seront pas pourvues.



Voici les connexions que nous venons de créer.

Afin que le node *voronoifracture1* transmette l'attribut **active** aux débris qu'il a produits, aller dans l'onglet **Attributes**, activer l'option *Copy Cell Point Attributes*, et rentrer : **active** dans le champ *To Piece Points*.



Séparer la géométrie en deux groupes :

Créer un node *Delete* en dessous du *Vornoifracture1*.

Mettre le paramètre *Opération* sur : **Delete Non-Selected**.

Dans l'onglet **Number**, régler le paramètre *Opération* sur : **Delete By Expression**.

Écrire dans *Filter Expression* : **\$ACTIVE == 1**

Renommer le node : **delete_passive**

Placer en dessous un node *Null*, puis le renommer : **OUT ACTIVE**

L'expression va sélectionner toutes les primitives ayant un attribut *active* égale à 1. Les polygones qui ne sont pas sélectionnés seront supprimés.

Dupliquer le node *delete_passive*.

Mettre le paramètre *Opération* sur : **Delete Selected**.

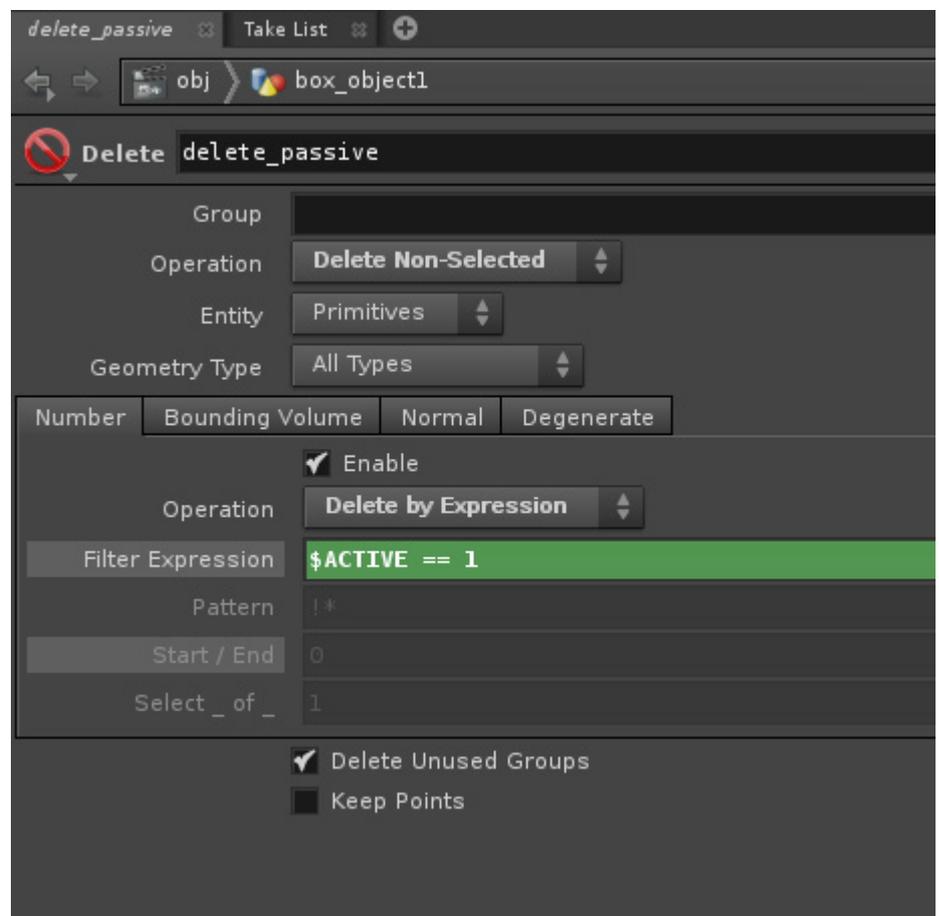
Renommer le node :

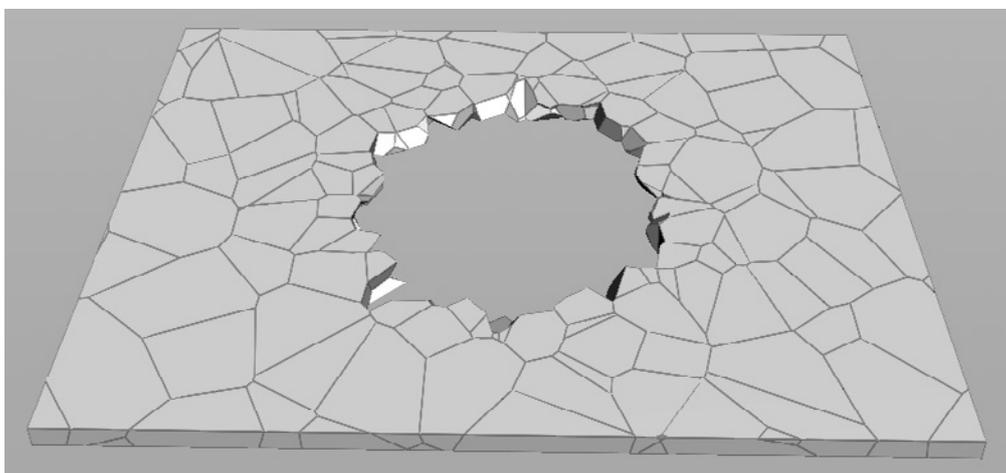
delete_active.

Placer en dessous un node

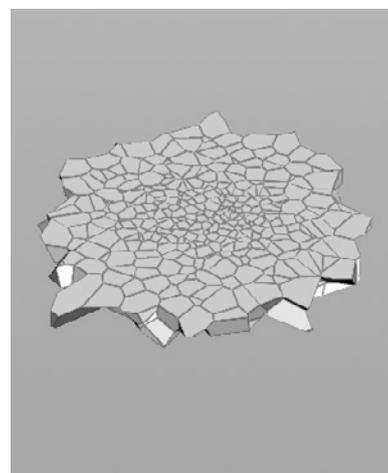
Null, et le renommer : **OUT**

PASSIVE.

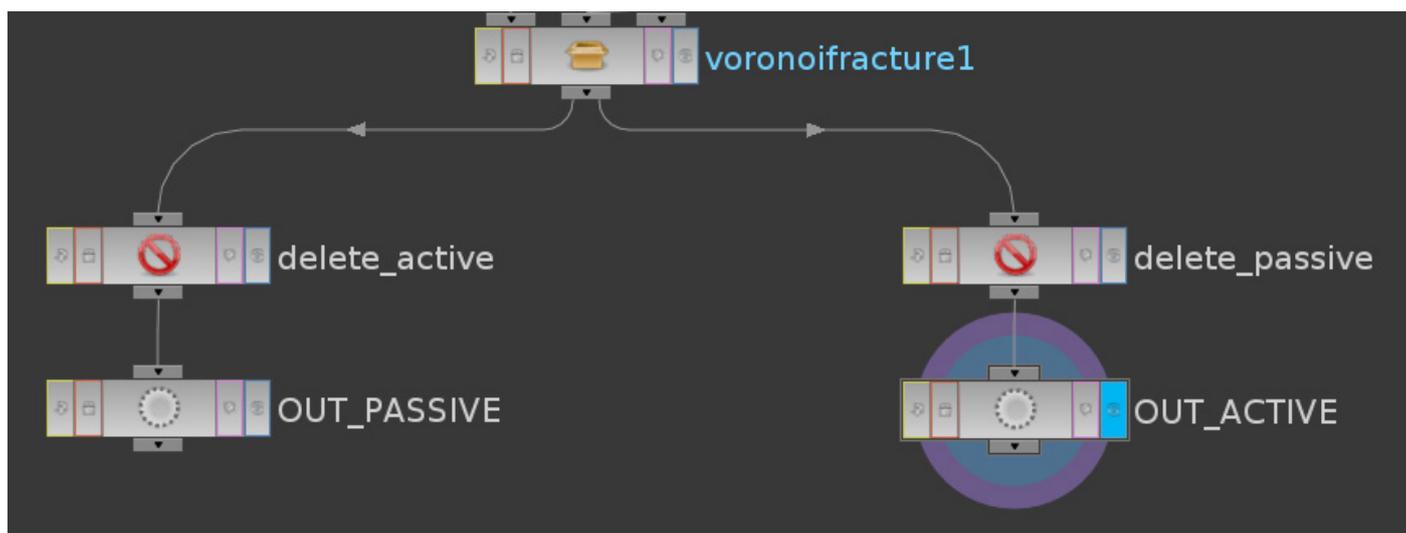




OUT PASSIVE



OUT ACTIVE



La géométrie OUT ACTIVE s'affaissera, alors que l'OUT PASSIVE sera statique. Dans l'état actuel, les débris tomberont sans que rien ne les arrête. Nous allons donc créer un mesh de collision en dessous du ground pour les recevoir.

Créer le mesh de collision :

Dans le node *box_object1*, créer une box. La renommer **COLLISION**.

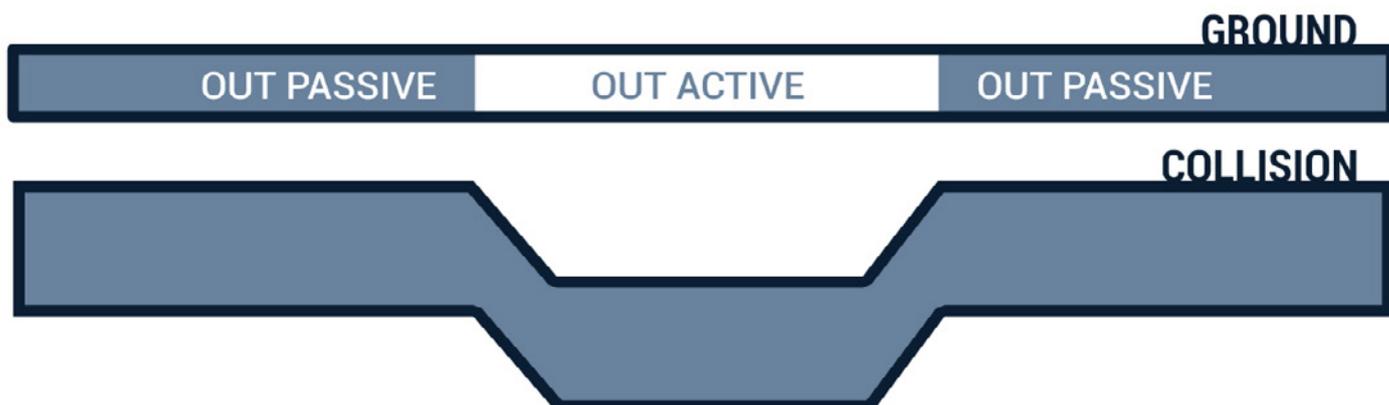
Mettre le paramètre *Primitive Type* sur : **Polygon Mesh**.

Régler son *Size* à : **19 0.6 26**

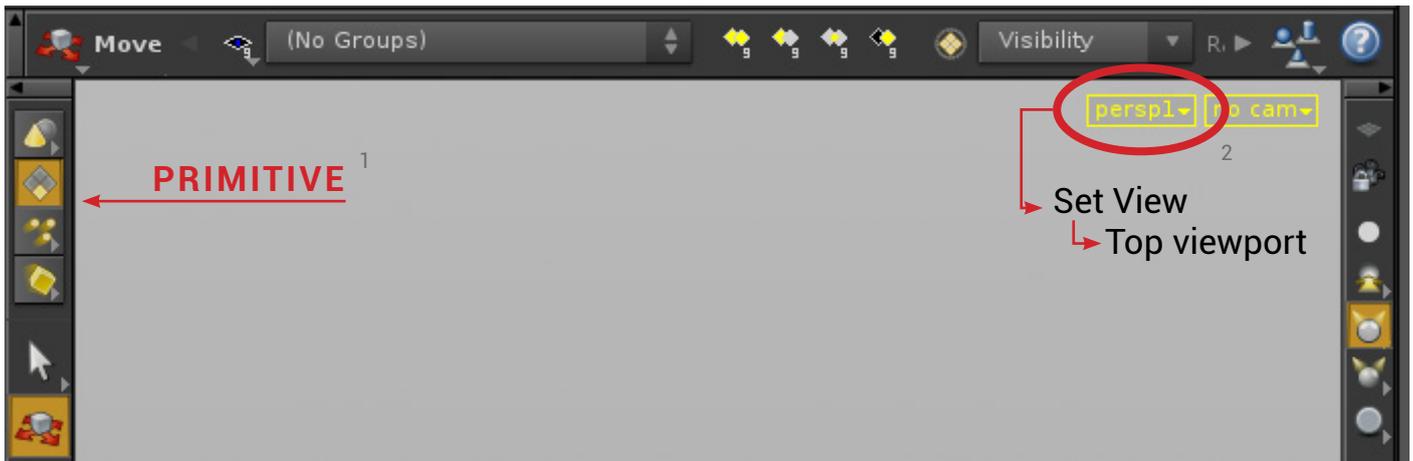
Center à : **0 -0.7 0**

Axis Divisions à : **30 30 30**

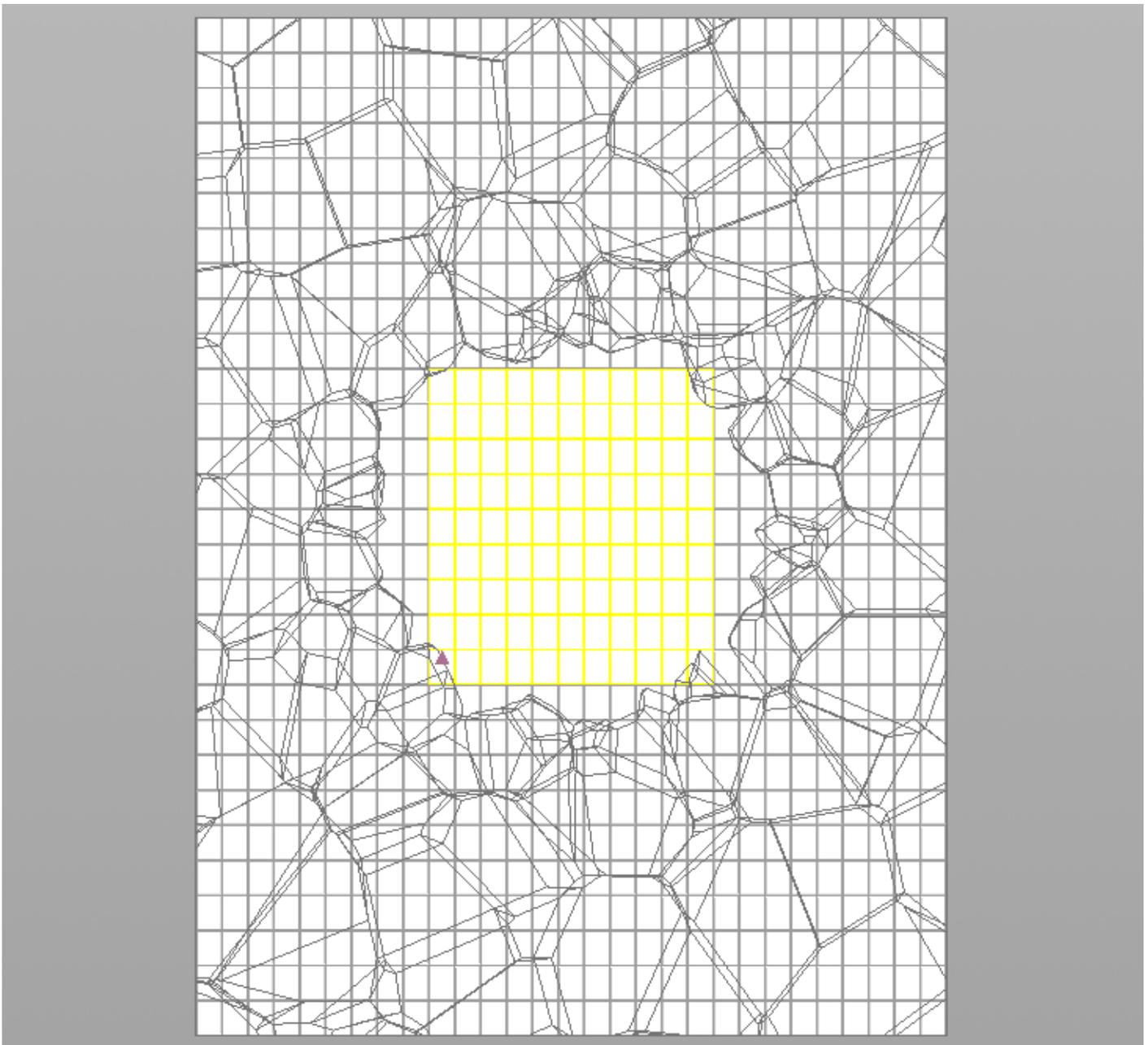
Notre but est de créer une cuvette sous la zone vide de la géométrie OUT PASSIVE.



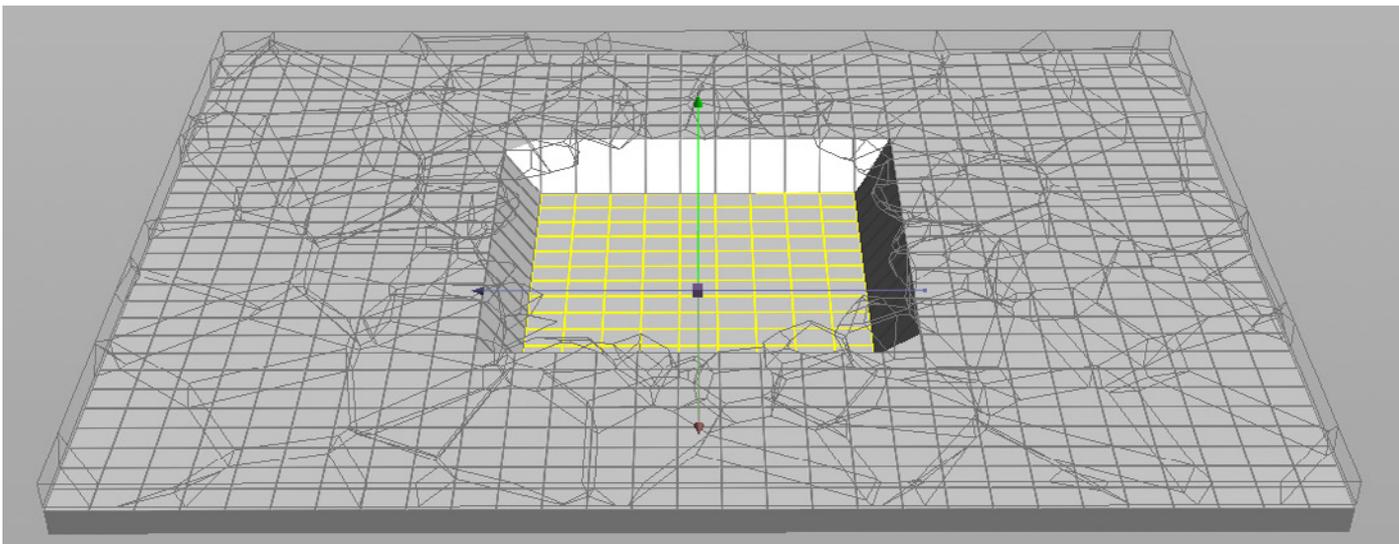
Se mettre en mode de sélection **Primitive**¹.
Dans la Scène View, se mettre en vue **Top**².



Activer le **template** du node OUT PASSIVE.
Cliquer sur le node COLLISION, et le mettre en mode **Display**.
Puis sélectionner les polygones placés dans la zone sans fracture (ici en jaune).



Revenir dans la vue perspective, et pousser la sélection sur l'axe -Y.



Créer en dessous du node *Edit1* un node *Null*, le renommer : **OUT MESH COLLISION**.

↑ Note :

Houdini dispose de différents solvers pour détecter les collisions. Parmi eux nous allons utiliser Bullet, il a l'avantage d'être rapide à calculer, mais il crée des erreurs sur les géométries concaves (creuses). Car il utilise un système de simplification des meshes appelé Tétraédrisation qui supprime toute forme concave.

Si nous lui donnons à simuler notre cuve il ignore le creux. C'est pourquoi je vais présenter deux techniques pour pallier ce défaut.

Rendre collisionable une forme creuse avec Bullet :

De la même manière qu'avec notre géométrie GROUND, nous allons fracturer en diverses pièces le mesh COLLISION, à la différence que nous souhaitons qu'il reste solidaire.

Nous le rendrons collisionable grâce au node *Fractured Object* du shelf **Rigid Bodies**. Ce node, va de manière automatique, créer un mesh de collision pour chaque pièce issue de la fracturation. Ces dernières devraient en principe avoir des formes rectilignes, supprimant donc tout problème de concavité.

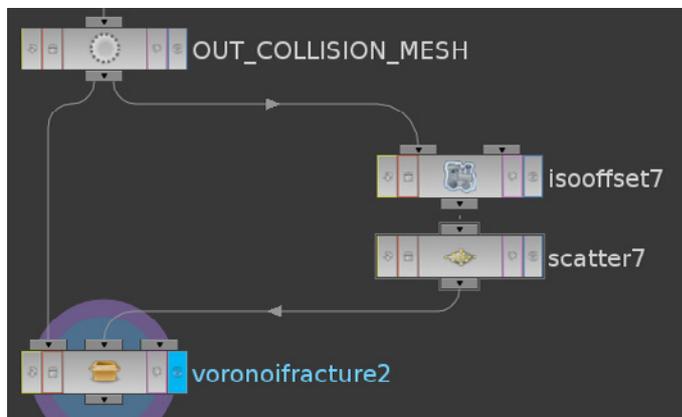
Première technique :

En dessous du node OUT COLLISION MESH, créer un *IsoOffset*.

Régler son paramètre *Uniform Sampling* à : **100**

Relier son output à un node *Scatter*.

Créer un node *Voronoi Fracture*. Placer l'output du node *scatter7* dans son second input, puis mettre l'output du node OUT COLLISION MESH dans son premier input.



Aperçu de la Tétraèdrisation :

La géométrie est prête pour la simulation, mais pour prévenir tout bug il est préférable de visualiser le mesh de collision créé par Bullet. Grâce à un node *for each* nous allons transformer séparément chaque pièce en Tétraèdre.

Relier l'output du node *voronoifracture2* au premier input d'un node *For Each*.

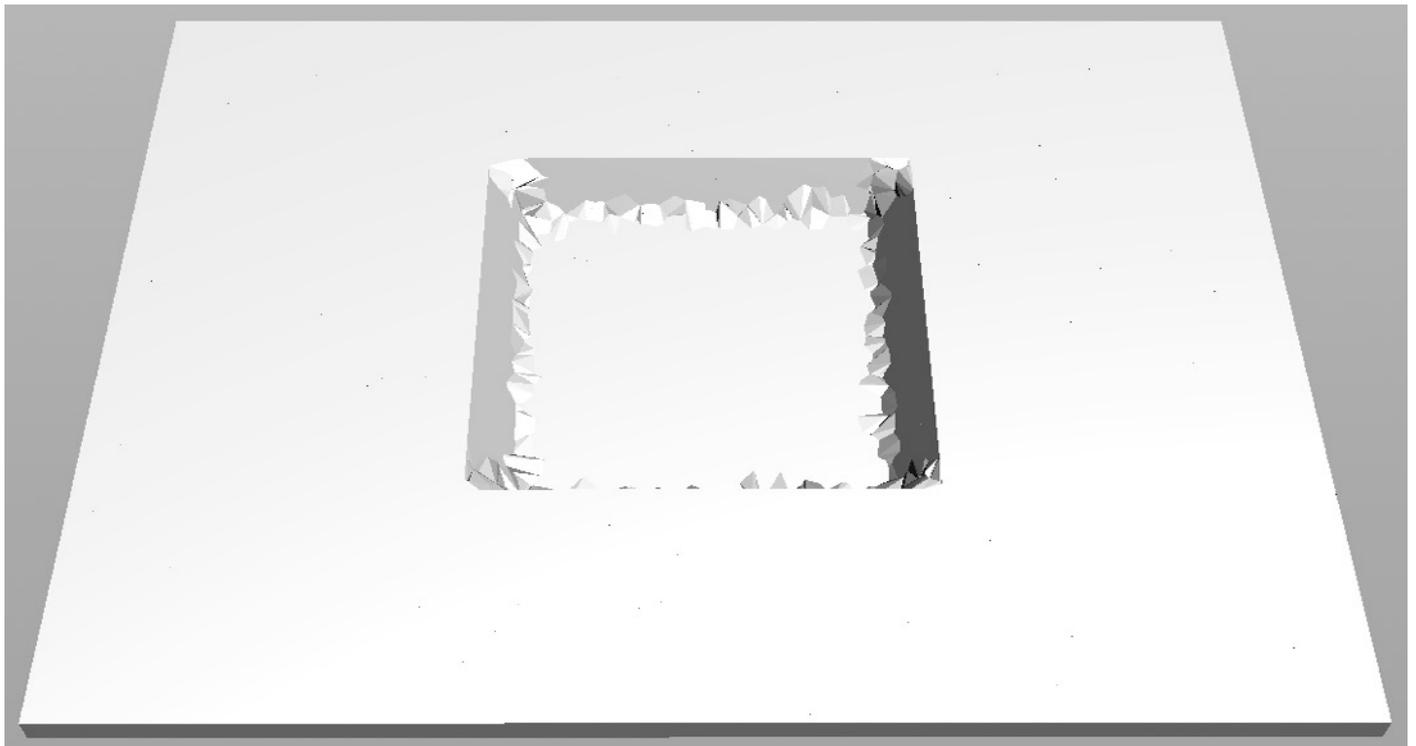
Mettre le paramètre *For* sur : **Each Attribut Value**.

Enfin écrire dans le champ *Attribute* : **name**.

Ainsi, il sélectionnera tour à tour chaque pièce du mesh COLLISION, afin de leurs appliquer séparément les connections que nous allons créer dans son network.

Dans le node *foreach1*, brancher en dessous du node *each1* un node *Tetrahedralize*.

Puis sortir du node.



On peut constater que certaines fractures sur la partie basse du creux avaient une forme convexe, car suite à la tétraèdrisation les angles se sont transformés en surface linéaire. Malgré ces petites erreurs, nous avons la forme globale de la cuvette, pour notre simulation une plus grande précision serait inutile. Dans le cas contraire, nous utiliserions la seconde technique ou le solver RBD.

Seconde technique :

Elle consiste à mesurer la convexité du mesh afin de créer plus de fractures dans les zones anguleuses. Si vous avez suivi la première technique, supprimer tous les nodes après le OUT COLLISION MESH.

En dessous du node *OUT COLLISION MESH*, créer un *Attribut creat*.

Écrire dans le champ du paramètre *Name* : **curvature**.

Créer en dessous un node *Measure*.

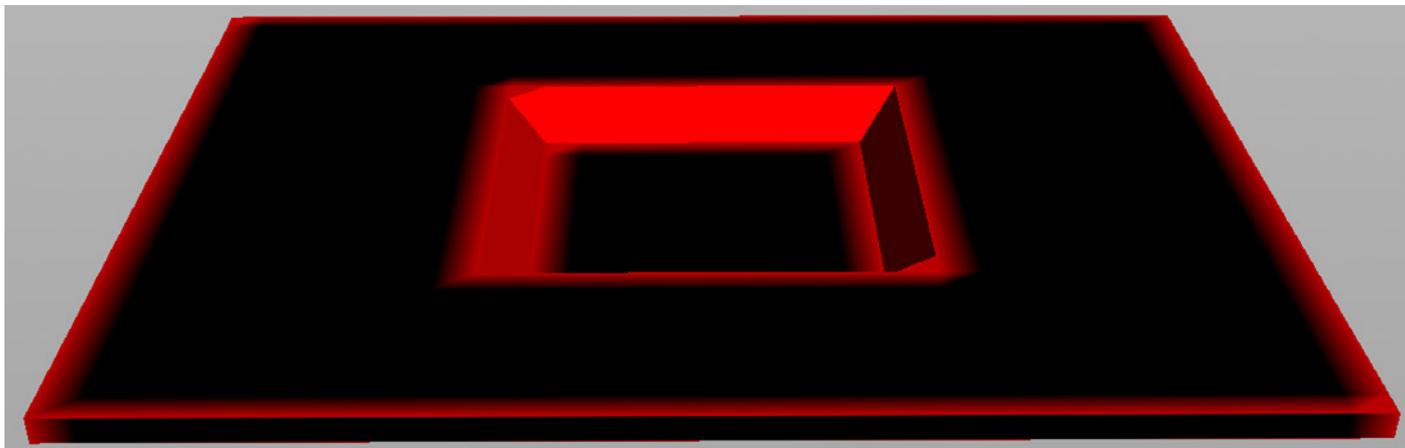
Mettre le paramètre *Type* sur : **Curvature**.

Placer en dessous du node *Measure* un node *Point*.

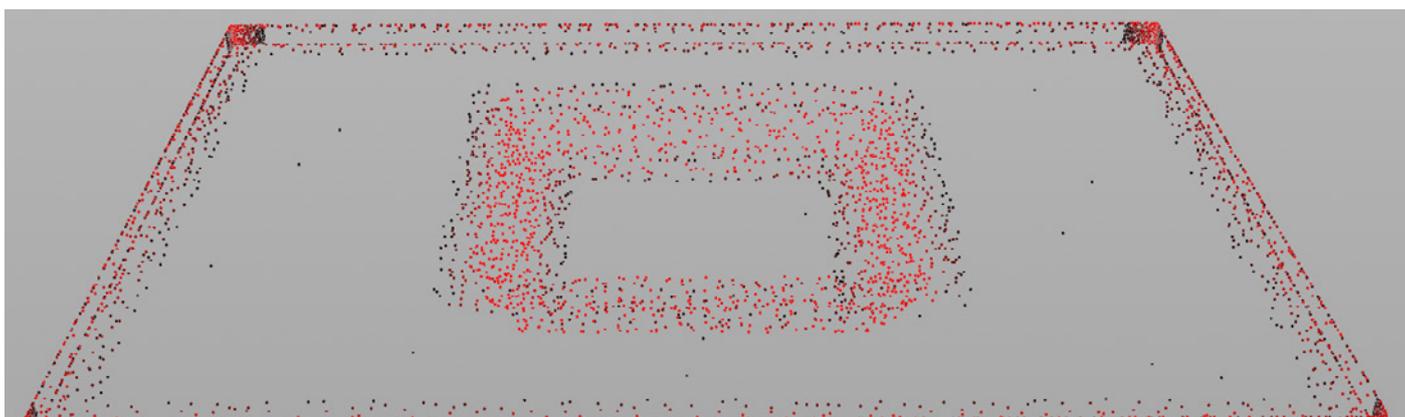
Régler le paramètre *Keep Color* sur **Add Color**. Faire un **Clic-Droit** sur *Color* et sélectionner : **Delete Channels**.

Mettre tous les channels à **0**, puis écrire dans le channel rouge (le premier) :
fit(\$CURVATURE, 0, 0.2, 0, 1)

Afin d'offrir une interprétation correcte des mesures de courbure par la couleur, nous utilisons la fonction `fit()`. Elle va remapper une distance entre deux nombres vers une autre. Ici nous transformons les valeurs de courbure qui vont de 0 à 0.2, à des valeurs de couleur allant de 0 à 1.



Dans le but de répartir plus de points dans les zones colorées et d'y créer plus de fractures, placer en dessous du node point un node *scatter*.
Écrire dans le champ nommé *Alternate Attribute* : **Cd**.
Puis, régler le paramètre *Attribute Bias* à : **0.994**.



Créer un node *Voronoi Fracture*. Placer l'output du node *scatter7* dans son second input, puis mettre l'output du node *OUT COLLISION MESH* dans son premier input.

Pour avoir un aperçu de la tétrahédrisation se reporter à la sous partie : Aperçu de la Tétrahédrisation présent dans la partie **Première technique** (p.79).

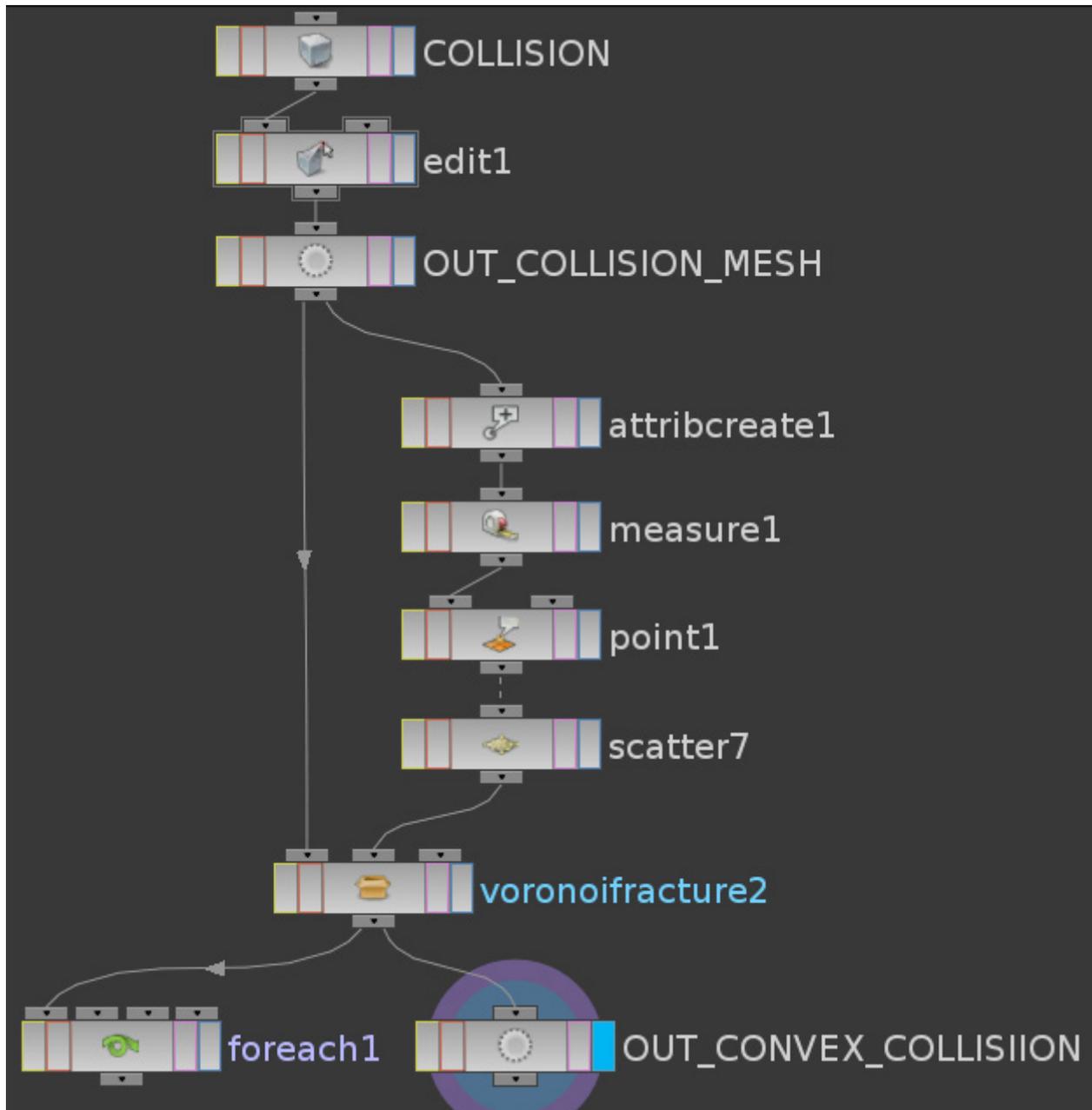


Ci-dessus le mesh suite à la tétrahédrisation.

Le résultat n'est pas flagrant sur un mesh aussi simple, mais cette technique peu aider dans bien des cas.

Si vous avez besoin d'une précision extrême, utilisez plutôt le solveur **RBD**. Qui se sert d'une représentation volumique des meshes pour calculer les collisions.

Placer un node *Null* sous le *voronoifracture2*, le renommer : **OUT CONVEX COLLISION**.



Le node *foreach1* sert à des fins de visualisation, le *Null OUT CONVEX COLLISION* quant à lui sert à exporter la géométrie vers le contexte des Dynamics.

Nous en avons terminé pour la première partie, nous allons à présent rendre dynamiques ces géométries.

SIMULER :

Nous avons trois éléments à simuler :

Le OUT ACTIVE : les débris qui s'affaisseront.

Le OUT PASSIVE : la partie du GROUND qui restera solidaire

Le OUT CONVEX COLLISION : la cuvette qui bloquera la chute des débris.

Nous allons les exporter dans des géométries séparées.

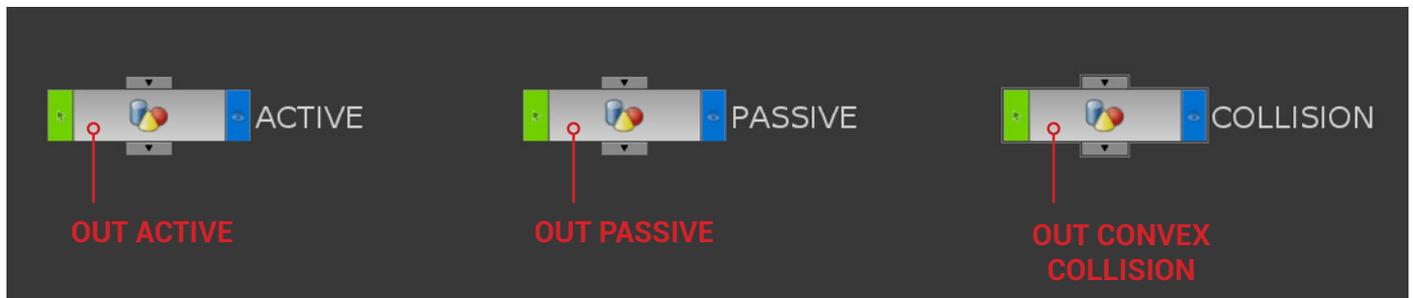
Dans la vue contextuelle **Scène**, créer un premier node *Géométrie*, le renommer **ACTIVE**, un second node *Géométrie*, le renommer **PASSIVE**, un troisième node *Géométrie*, le renommer **COLLISION**.

Dans ces trois nodes, supprimer le *file*, puis créer un *Object Merge*.

Dans la géométrie nommée **ACTIVE**, charger dans le paramètre *Object1* de son *Object Merge*, le *OUT ACTIVE* issue du node *box_object1*.

Dans celle nommée **PASSIVE**, charger le *OUT PASSIVE*.

Dans celle nommée **COLLISION**, charger le *OUT CONVEX COLLISION*.



Dans la vue contextuelle *Scène*, sélectionner le node *ACTIVE*, puis dans le *shelf Rigid Bodies* faire un **CTRL-Clic Gauche** sur l'icône *RBD Fracture*.

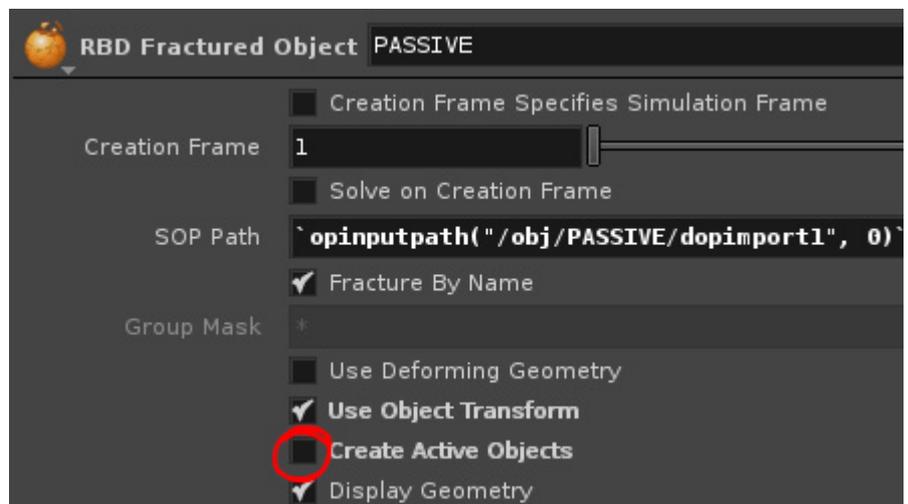


Faire de même pour les nodes *PASSIVE* et *COLLISION*. Puis désactiver le **display** des trois géométries. Suite à ces manipulations, un node *AutoDopNetwork* s'est créé, rentrer à l'intérieur et appuyer sur la **touche L** du clavier afin de réorganiser l'espace. C'est au sein de ce node que se configure la simulation.

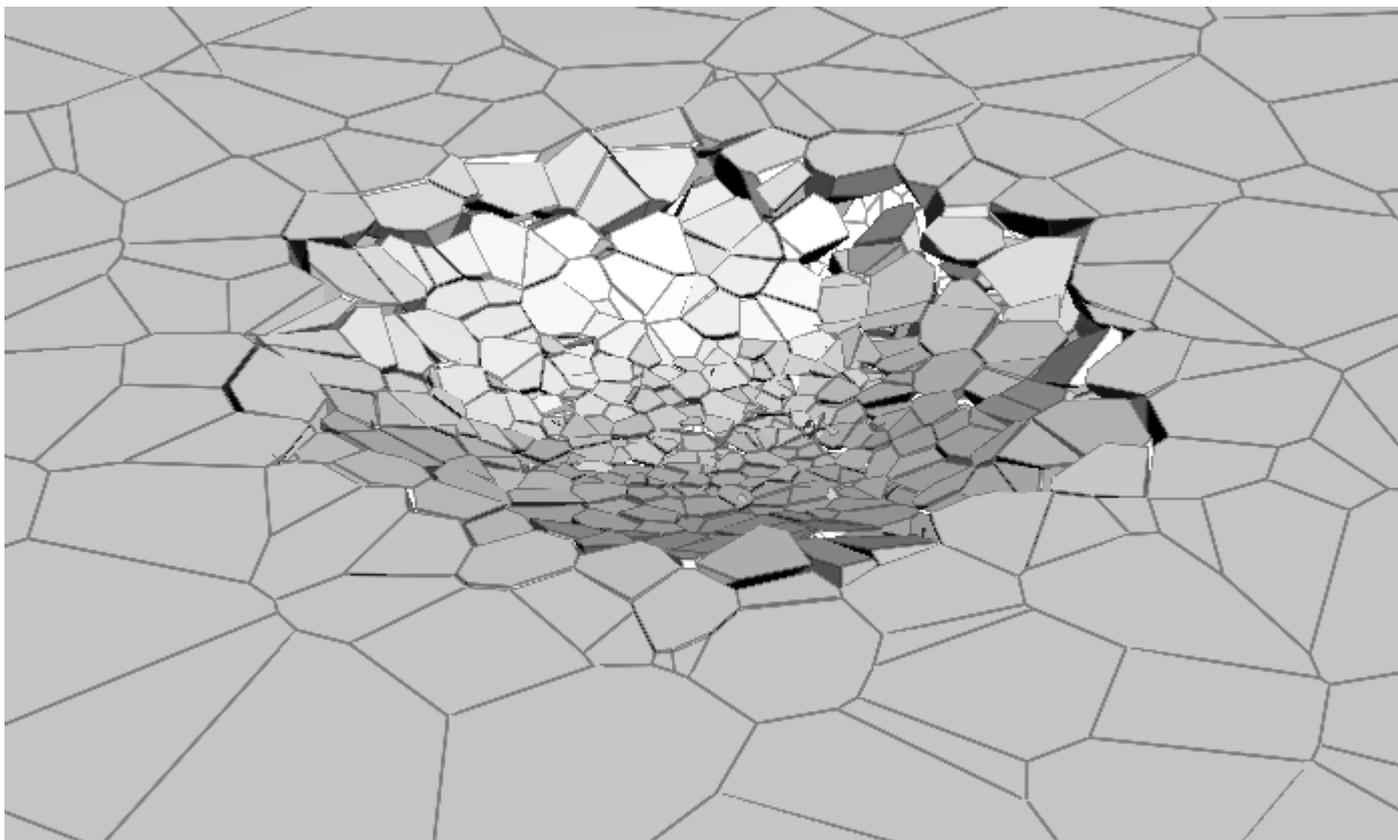
Si nous lançons la lecture, les trois géométries subiront l'effet de la gravité et se fractureront en de multiples pièces. Nous souhaitons que les géométries *PASSIVE* et *COLLISION* ne tombent pas et restent solitaires. Cliquer sur le node *PASSIVE* et désactiver l'option :

Creat Active Objects.

Faire de même pour le node *COLLISION*.



On peut constater qu'à présent, seules les pièces de la géométrie *ACTIVE* tombent.



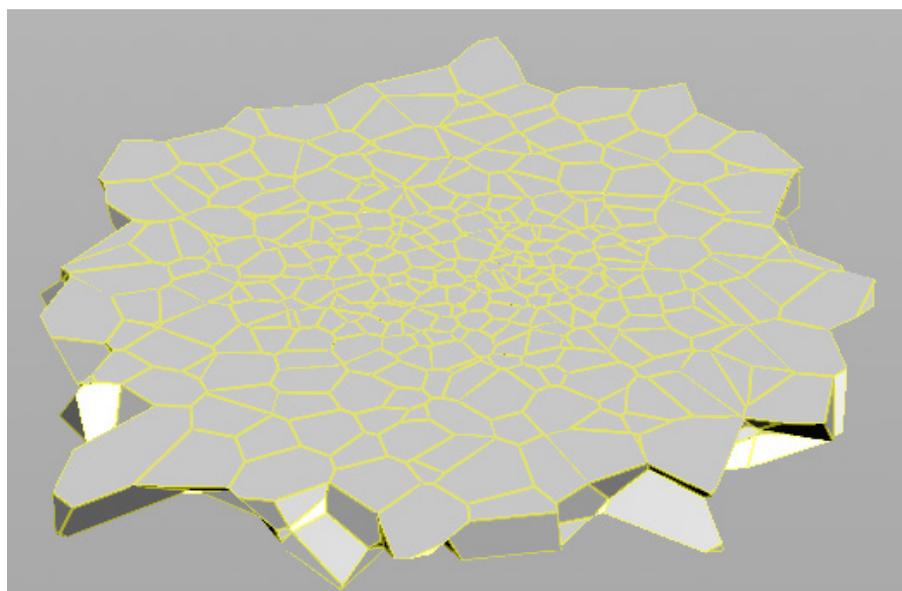
CONTRÔLER LE DÉTACHEMENT DES PIÈCES :

Toutes les pièces de la géométrie *ACTIVE* se détachent les unes des autres dès la première frame, grâce au node *Glue Constraint* nous allons les lier ensemble, en créant entre chacune d'elles une contrainte de glue. Nous aurons ensuite la liberté de les supprimer à notre guise.

Activer le **display** du node *ACTIVE*, afin qu'il soit le seul visible.

Dans la **Scène View**, se mettre en mode de sélection de **Primitive**, puis sélectionner l'ensemble du mesh.

Appuyer sur la touche **CTRL**, puis dans le shelf *Rigid Bodies*, cliquer sur l'outil *Glue Adjacent*.

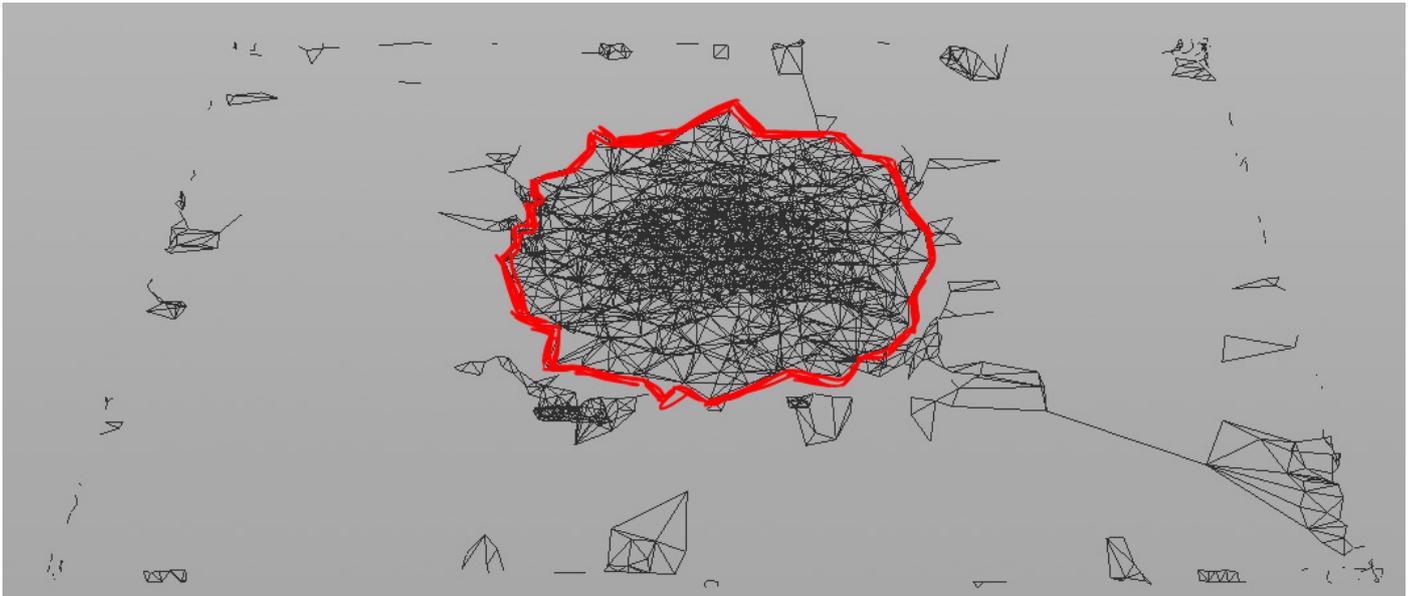


Ci-dessus la géométrie *ACTIVE* sélectionnée.

Un node *glue_piece100_to_piece99* s'est créé. Brancher son premier input sous le node *gravity*. Il va importer dans le Dopnet des contraintes de glue qui ont été créées dans une géométrie à l'extérieur de ce node. C'est grâce à cette géométrie que nous allons contrôler la manière dont se détacheront les contraintes.

Dans la vue contextuelle *Scène*, renommer le node *glue_piece100_to_piece99* en : **GLUE**, puis rentrer à l'intérieur.

Au centre de la scène se trouvent les contraintes de glue de la géométrie *ACTIVE* (entourées de rouge). Et gravitent autour d'elles d'autres contraintes, elles ont été créées par une erreur due à un conflit entre les noms des pièces issues des *Voronoi-Fracture* des géométries *ACTIVE* et *COLLISION*.



Pour y remédier, retourner dans la scène, rentrer dans la géométrie *box_object1*. Sélectionner le node *voronoifracture2* (celui du mesh *COLLISION*), puis dans son onglet **Attributs** écrire dans le champ *Name Prefix* : **piece_COLLISION**.

Enfin retourner dans la géométrie *glue_piece100_to_piece99*.

Il n'y a à présent plus que les contraintes de la géométrie *ACTIVE* qui apparaissent.

Dans le but de faire tomber progressivement les pièces du centre vers l'extérieur, nous allons créer une sphère qui va colorer les contraintes que nous voulons supprimer. Cette sphère contrôlera le détachement des débris.

Créer une sphère, puis brancher dans son output un node *point* (premier input).

Mettre le paramètre *Keep Color* sur *Add Color*, faire un **Clic-Droit** sur *Color* et sélectionner : **Delete Channels**.

Enfin pour obtenir du rouge régler les channels de couleur comme suit : **1 0 0**

Dupliquer le node, et le brancher à l'output du *gluepieces1*.

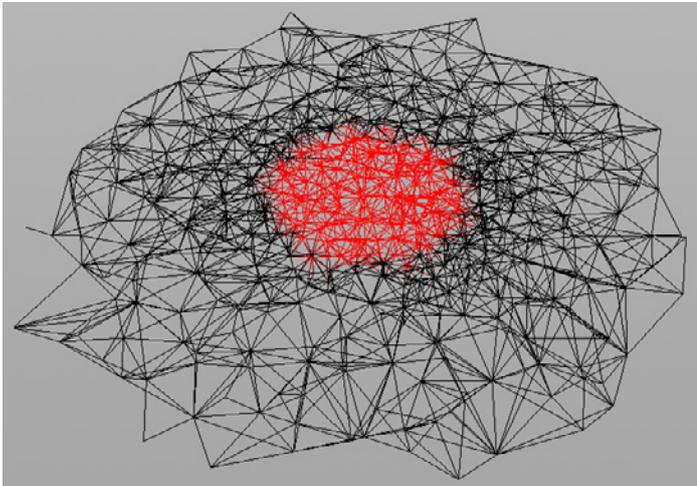
Puis régler les channels de couleur de sorte à avoir du noir : **0 0 0**

Créer un node *Attribut transfer*, mettre dans son premier input le *point2*, et dans son second input le *point1*.

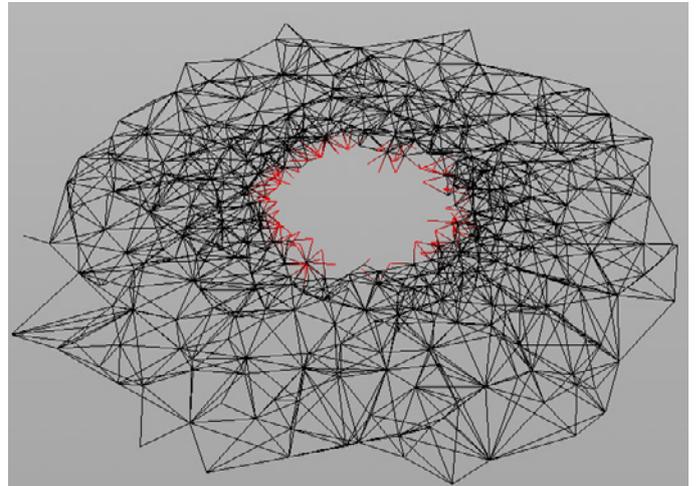
Dans l'onglet **Condition** du node *attributtransfer1*, baisser le paramètre *Distance Threshold* à une valeur de : **1.5**.

Nous allons animer ce paramètre afin de répandre la couleur sur l'ensemble des contraintes.

Mais avant, créer un node *Delete* sous le node *attribtransfer1*.
Dans son onglet **Number**, régler le paramètre *Operation* sur : *Delet By Expression*.
Puis écrire dans le champ *Filter Expression* : $\$CR == 1$.
L'expression signifie :
La valeur du **Channel Rouge** est égale à 1.
Le node supprimera ainsi toute primitive rouge.

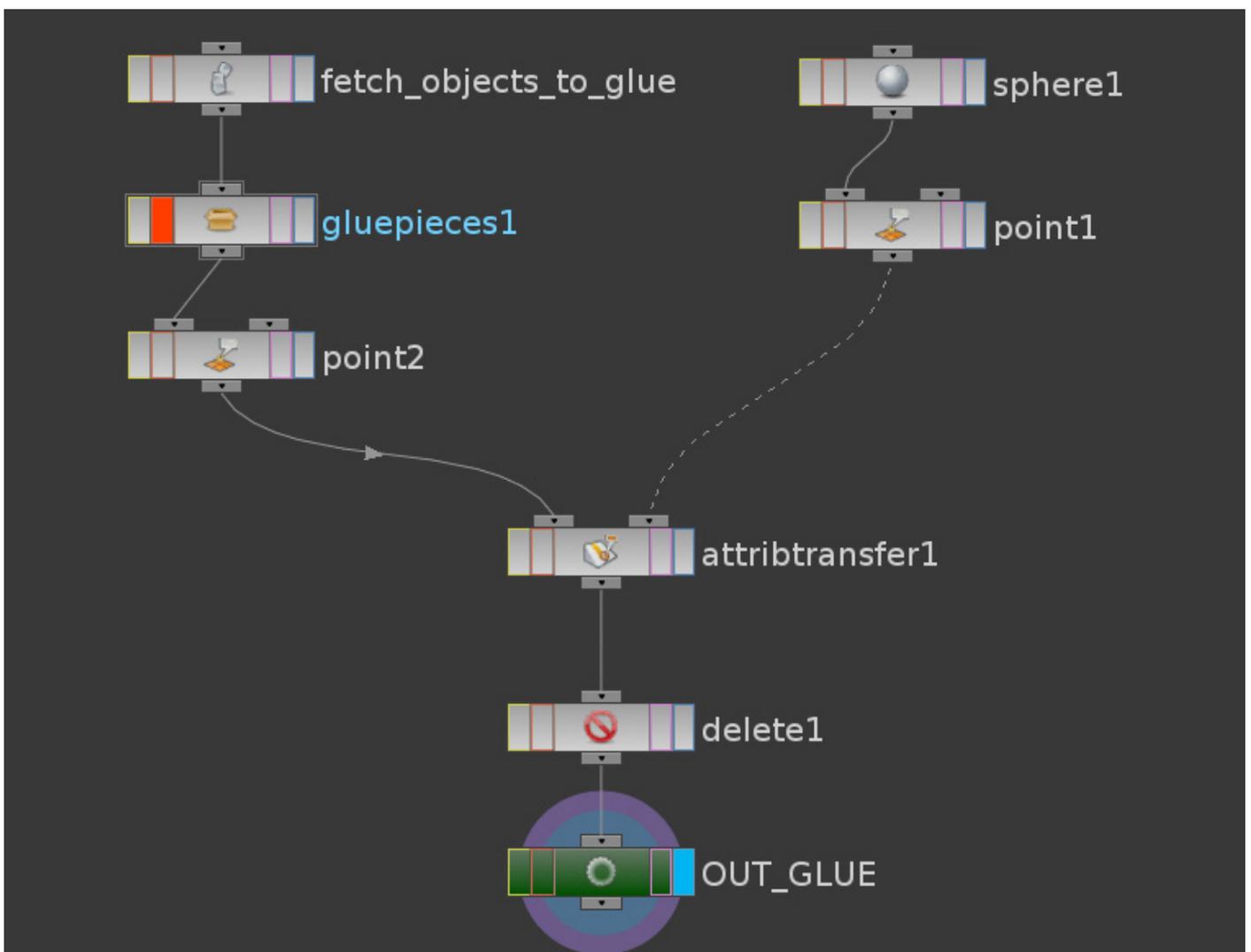


Les contraintes sont rouge suite au transfert de couleur de la sphère.



Elles sont supprimées par le node Delet.

Brancher à l'output du node *delete1* un node *Null*, le renommer : **OUT GLUE**.



Retourner dans la vue contextuelle **Scène**, puis rentrer dans le node *AutoDopNetwork*.
Cliquer sur le node *glue_piece100_to_piece99*, sélectionner dans le paramètre *Constraint Network* le *Null OUT_GLUE* issu du node *GLUE*.

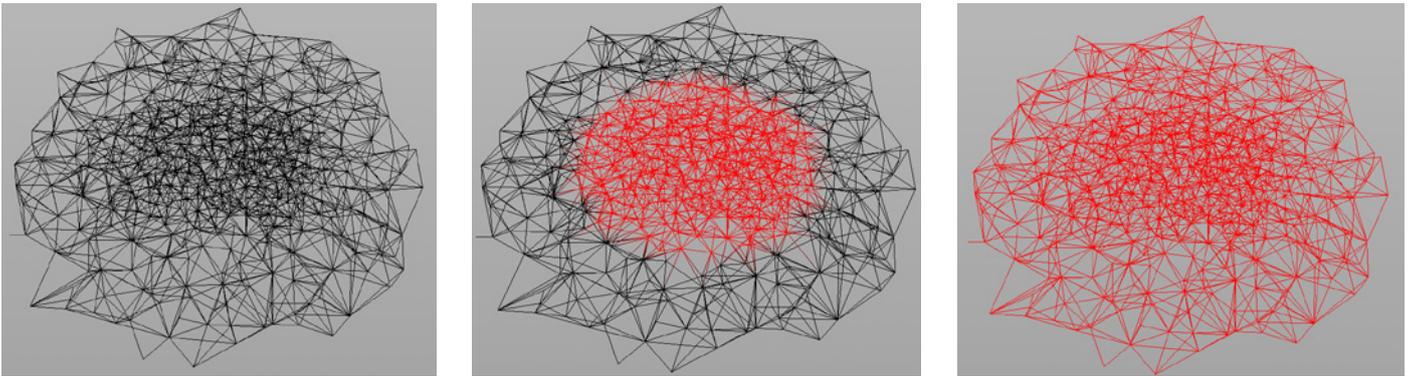
Ainsi le Dopnet chargera correctement la nouvelle géométrie de contrainte.

Puis faire un **Clic-Droit** sur le paramètre *Overwrite with SOP*, et sélectionner : **Delete Channels**.
Le Glue Network réactualisera à chaque frame la géométrie de contrainte, lui permettant ainsi d'être animée.

Dans la géométrie *GLUE*, nous pouvons à présent jouer avec le paramètre *Distance Threshold* et l'animer afin de créer différentes animations de fracture.

À la frame **1** mettre le paramètre à une valeur de **0**, faire un **alt Clic-Droit** sur le nom *Distance Threshold* afin de placer une clef d'animation.

À la frame **24** mettre le paramètre à **8**, de sorte à recouvrir l'ensemble des contraintes de rouge, puis mettre une clef.



Ainsi au fur et à mesure que l'animation avancera, les contraintes se supprimeront depuis le centre vers l'extérieur.

EMPÊCHER CERTAINES CONTRAINTES D'ÊTRE SUPPRIMÉES :

Nous allons peindre du vert sur les contraintes que l'on souhaite épargner.
Il y a deux manières de procéder, pour que le node *delete* ne supprime pas cette couleur.

La première :

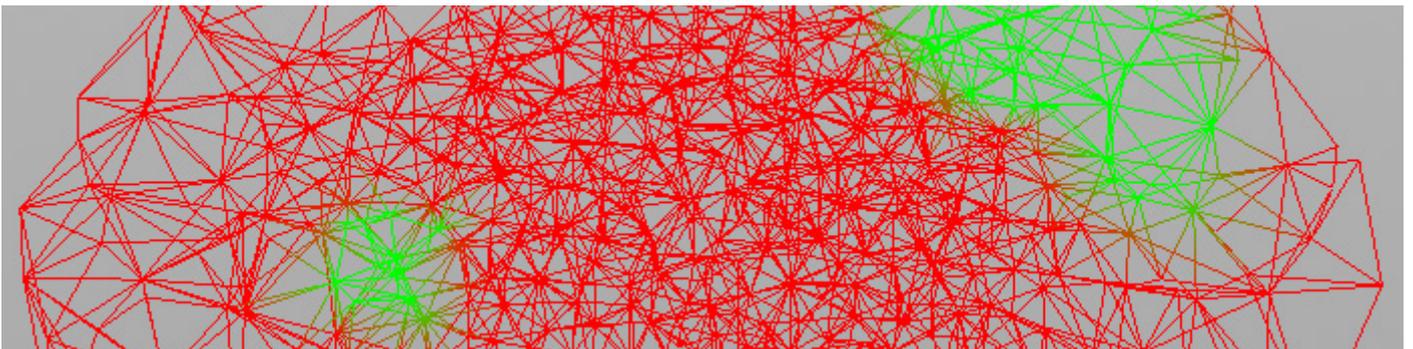
Cette solution est la plus simple. Mais peut ne pas convenir dans certains cas.

Créer un node *paint* entre les nodes *attribtransfer1* et *delete1*.

Dans l'onglet *Foreground Color* régler les channels comme suit : **0 1 0**

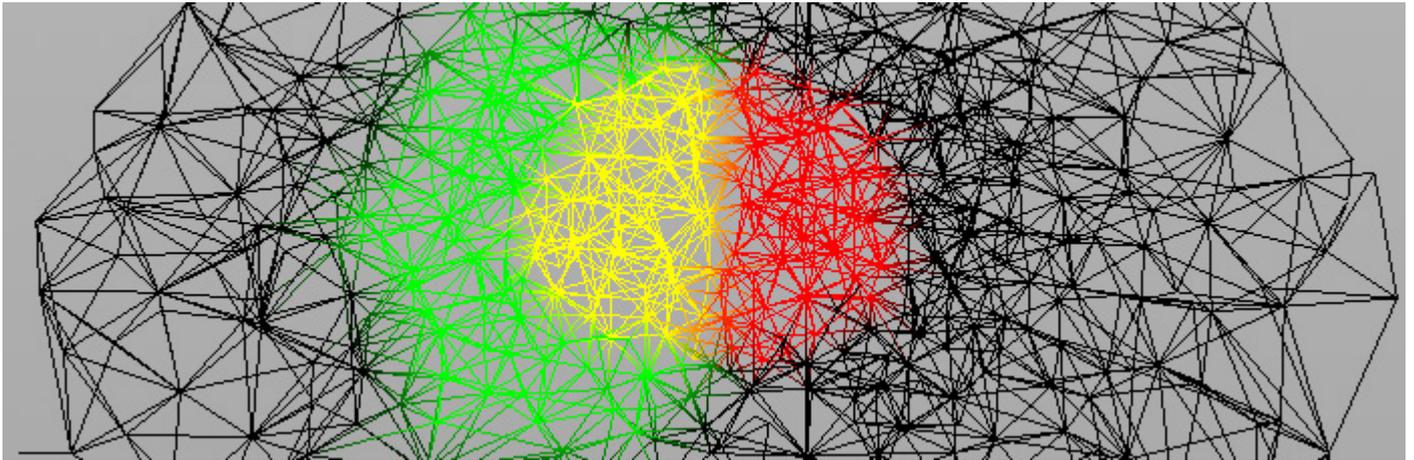
Avancer la Time-line jusqu'à l'image 24, afin que l'objet soit entièrement couvert de rouge.

Puis peindre les contraintes à préserver.



La Seconde :

En temps normal dès qu'une couleur recouvre une autre elles s'additionnent. C'est pourquoi si on peint le vert avant que le rouge vienne se superposer à lui, il deviendra jaune lorsqu'il sera recouvert.



Comme les valeurs du jaune sont égales à : **1 1 0**, et que l'expression du *delete* supprime les contraintes lorsque le **premier channel** à une valeur de **1**, ce qui aura été vert et qui est à présent jaune sera aussi effacé.

Pour remédier à ce problème, nous allons transférer les valeurs du vert dans un attribut, puis nous modifierons l'expression du *delete* de sorte qu'il n'efface pas les contraintes contenant cet attribut.

Créer un nouveau node *paint*, relier son input au output du node *attribtransfer1*.

Supprimer l'ancien node *paint*.

Dans l'onglet *Foreground Color* régler les channels comme suit :

0 1 0

Dès l'image **1** peindre les contraintes à préserver.

Placer en dessous du node *paint* un *attribut creat*.

Écrire dans son champ *Name* : **IGNORE**.

Puis dans le paramètre *Value* : **\$CG**.

Il créera donc un attribut **IGNORE** sur chaque point vert.

Créer en dessous un node *attribut*. Dans l'onglet *Point*, écrire dans le champ du paramètre *Delete Attributes* : **Cd**.

Ce node supprimera toute information de couleur sur l'objet, afin d'empêcher des conflits.

Placer entre les nodes *attribtransfer1* et *delete1* un nouvel *Attribut transfer*.

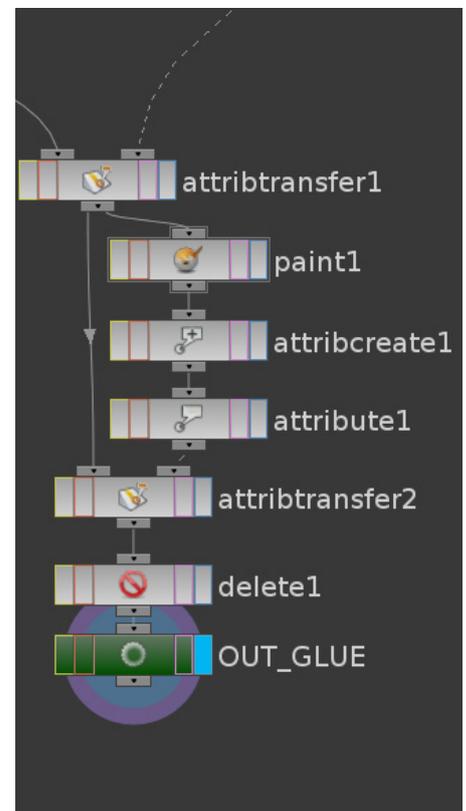
Brancher dans son second input l'output du node *attribut1*.

Il va transférer l'attribut **IGNORE** sur la géométrie colorée.

Dans le node *delete*, écrire dans le champ *Filter Expression* :

\$CR == 1 && \$IGNORE == 0

Ce qui équivaut à : si le channel rouge à une valeur de 1 et que l'attribut **IGNORE** à une valeur de 0.



SOP : RÉPANDRE UNE COULEUR SUR LA SURFACE D'UNE GÉOMÉTRIE - LE POINT CLOUD (H 12)

Le principe est de créer un nuage de points blanc à la surface de l'objet, de colorer un ou plusieurs d'entre eux en rouge, puis via le node point cloud, de permettre aux points rouges de communiquer leurs teintes à ceux qui les avoisinent. À chaque frame, les nouveaux points rouges transmettent à leurs tours leurs couleurs. On transfère ensuite cette couleur sur la géométrie, puis on la réutilise afin d'effectuer diverses opérations, telles que : supprimer des polygones, appliquer une texture, etc.

Fichier joint : PointCloud.hip

ENGENDRER LE NUAGE DE POINTS :

Nous allons utiliser pour ce tutoriel une simple sphère polygonale.

Créer la Sphère :

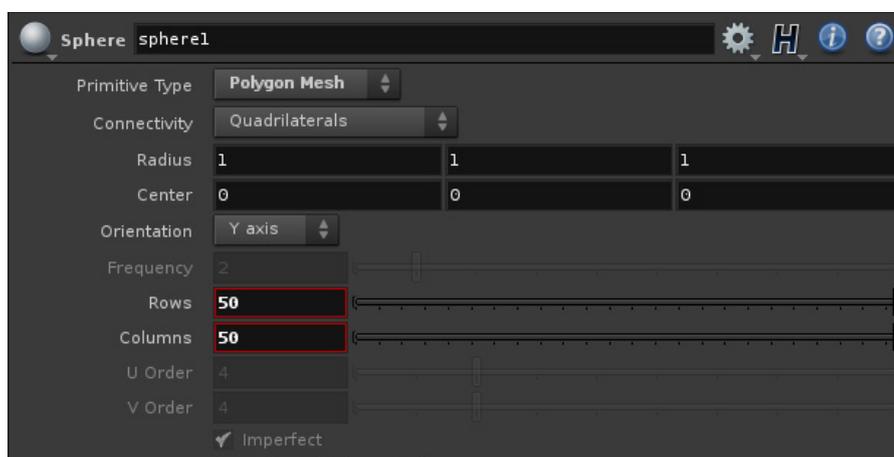
Dans le shelf *Creat* faire un **Ctrl + Clic Gauche** sur l'option *Sphere*.



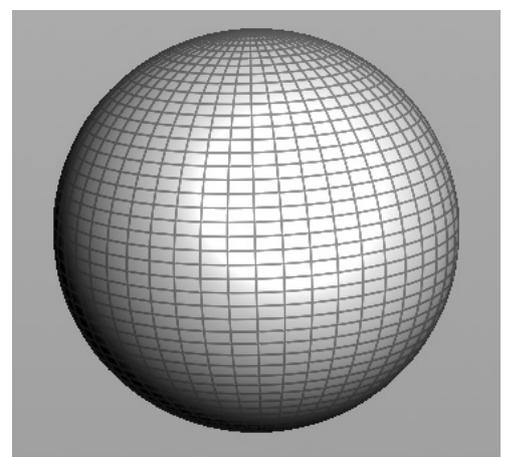
Dans le *Network Editor* double cliquer sur le node *sphere_object1* pour rentrer à l'intérieur.

Dans le *Parameter Editor* mettre le paramètre *Primitive Type* sur *Polygon Mesh* afin que la sphère soit constituée de faces à 4 côtés.

Puis augmenter le *Rows* et le *Columns* à **50**, de façon à avoir une densité de polygone intéressante pour mettre en place l'effet.



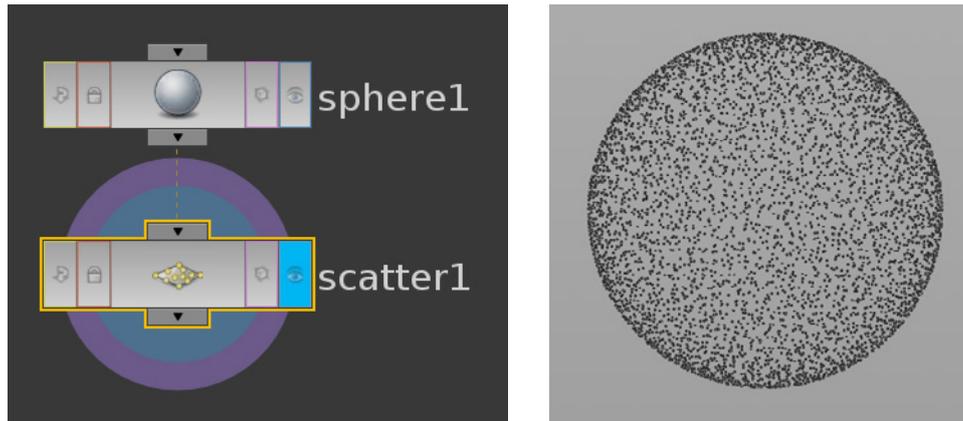
Les options présentes dans le *Parameter Editor*.



La sphère une fois convertie.

Nous allons à présent répartir des points à sa surface :

Créer un node *scatter*, puis le relier à l'Output du node *sphere1*.



SÉLECTIONNER LE POINT D'ORIGINE DE LA TEXTURE :

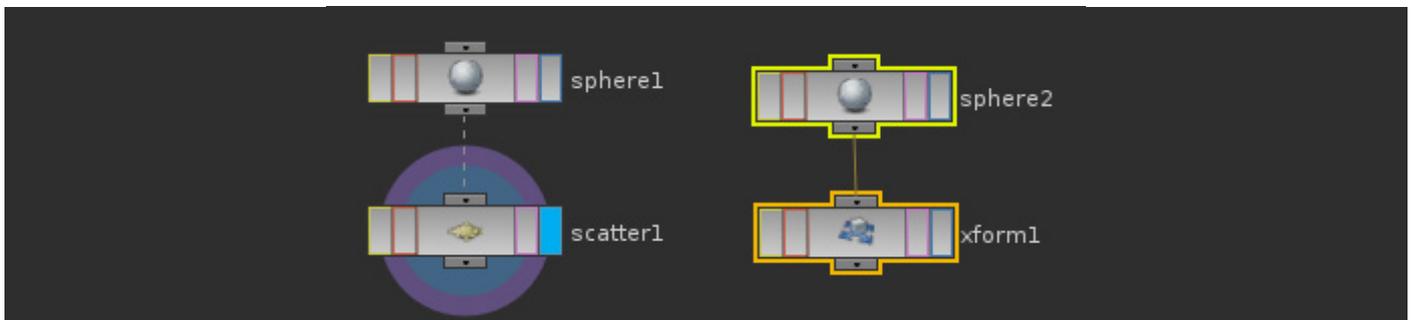
Nous allons créer une seconde sphère, elle nous servira à sélectionner les points qui diffuseront la couleur :

Cliquer dans une zone vide du *Network Editor* puis presser la touche **TAB**, enfin entrer : *sphere*.

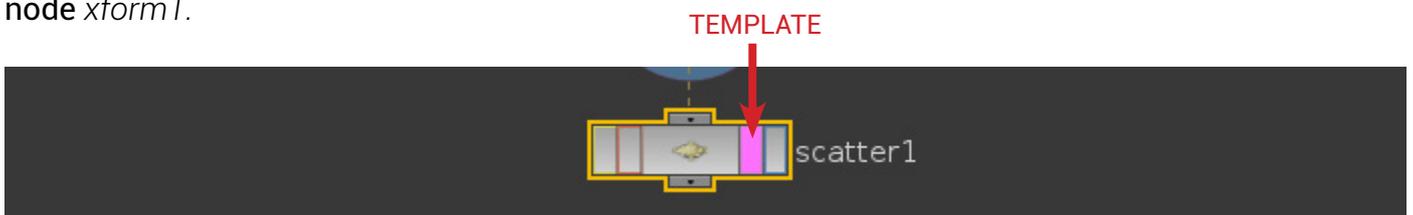
Cette sphère n'est pas créée via le shelf *Creat* car elle apparaîtrait en dehors du contexte de notre géométrie actuelle.

Créer un node *Transform* puis le relier à l'Output du node *sphere2*.

Il nous servira à modifier la position de la sphère dans l'espace 3D.

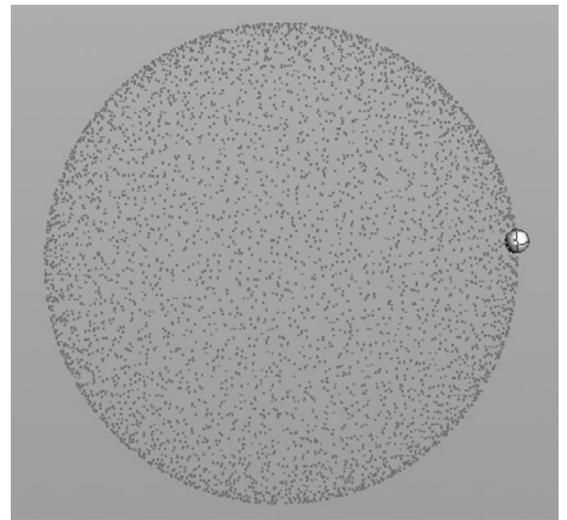


Activer le **template** du node *scatter1*, afin que le scatter soit visible lorsque l'on manipulera le node *xform1*.

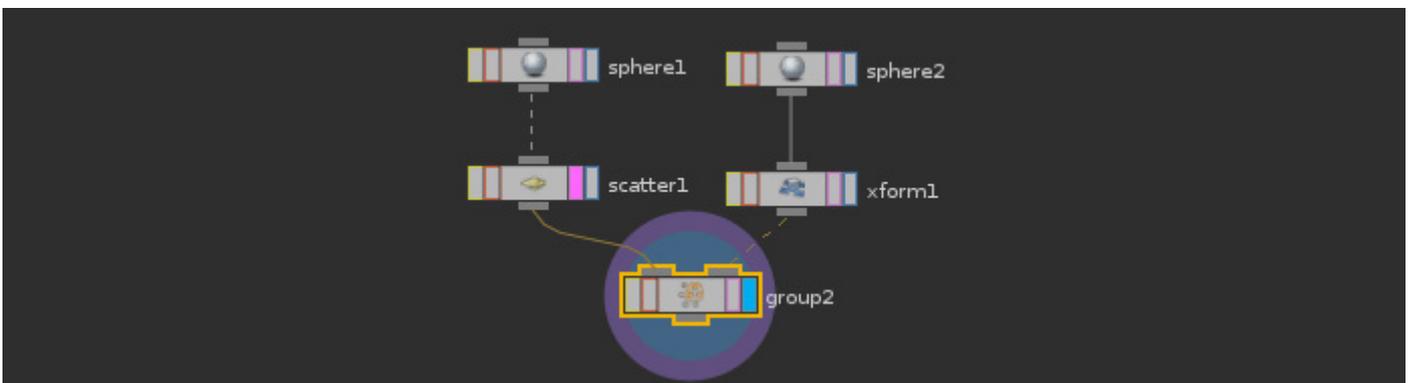


Baisser le *Uniform Scale* du node *xform1* à : **0.05**

Puis mettre le *Translate X* du node *xform1* à **1** afin de décaler la *sphère2* à la limite du nuage de point de la *sphère1* (voir page suivante)



Enfin relier le node *scatter 1* et le node *xform1* par un node *Group Geometry* : mettre dans son input gauche le node *Scatter1*, puis dans son input droit le node *xform1*.



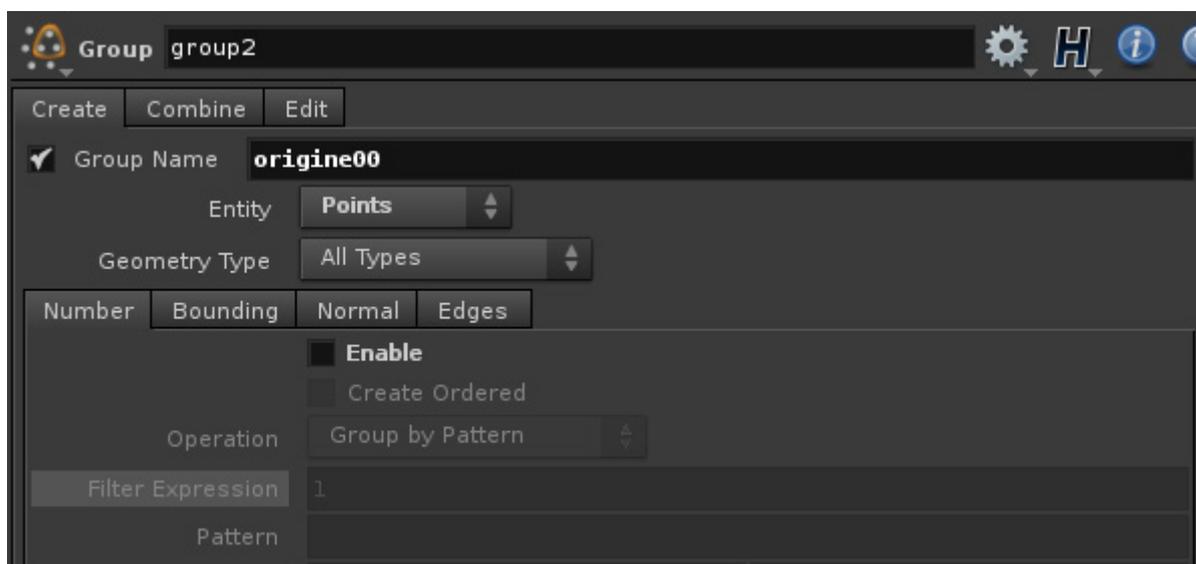
L'input Gauche correspond à la géométrie dans laquelle on va sélectionner des éléments.

L'input Droit correspond à la géométrie qui va servir d'espace de sélection.

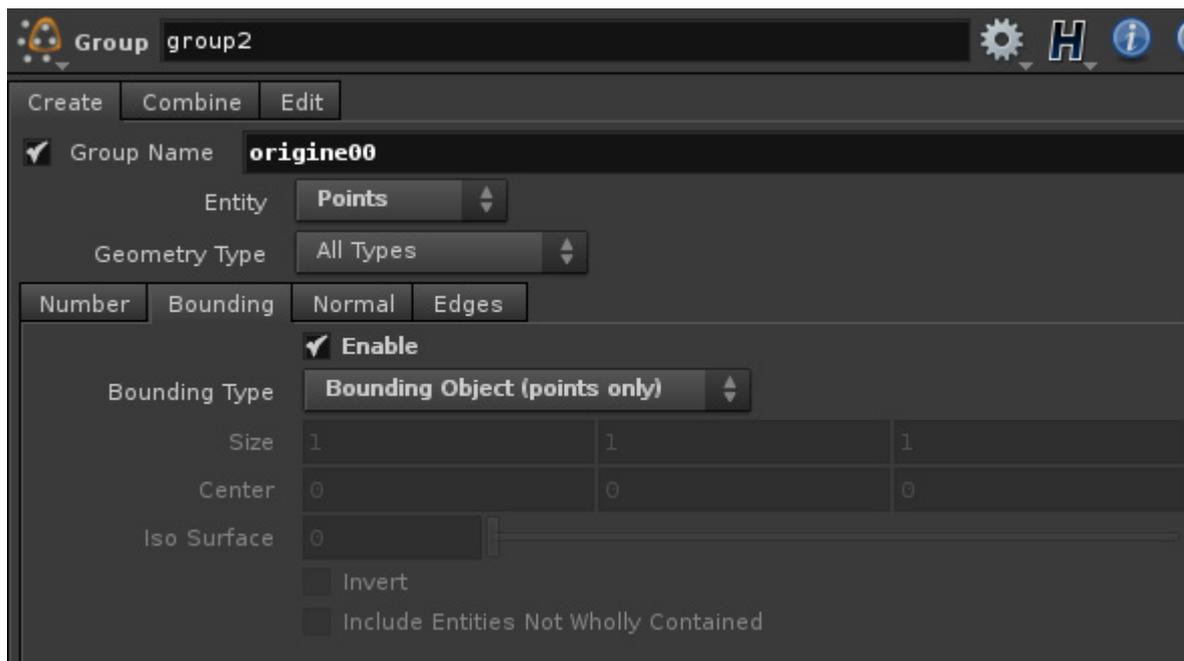
Écrire dans le *Group Name* du node *group2* : **origine00**

Régler le *Entity* du node *group2* sur : **Points**, car nous souhaitons sélectionner les points générés par le *scatter*.

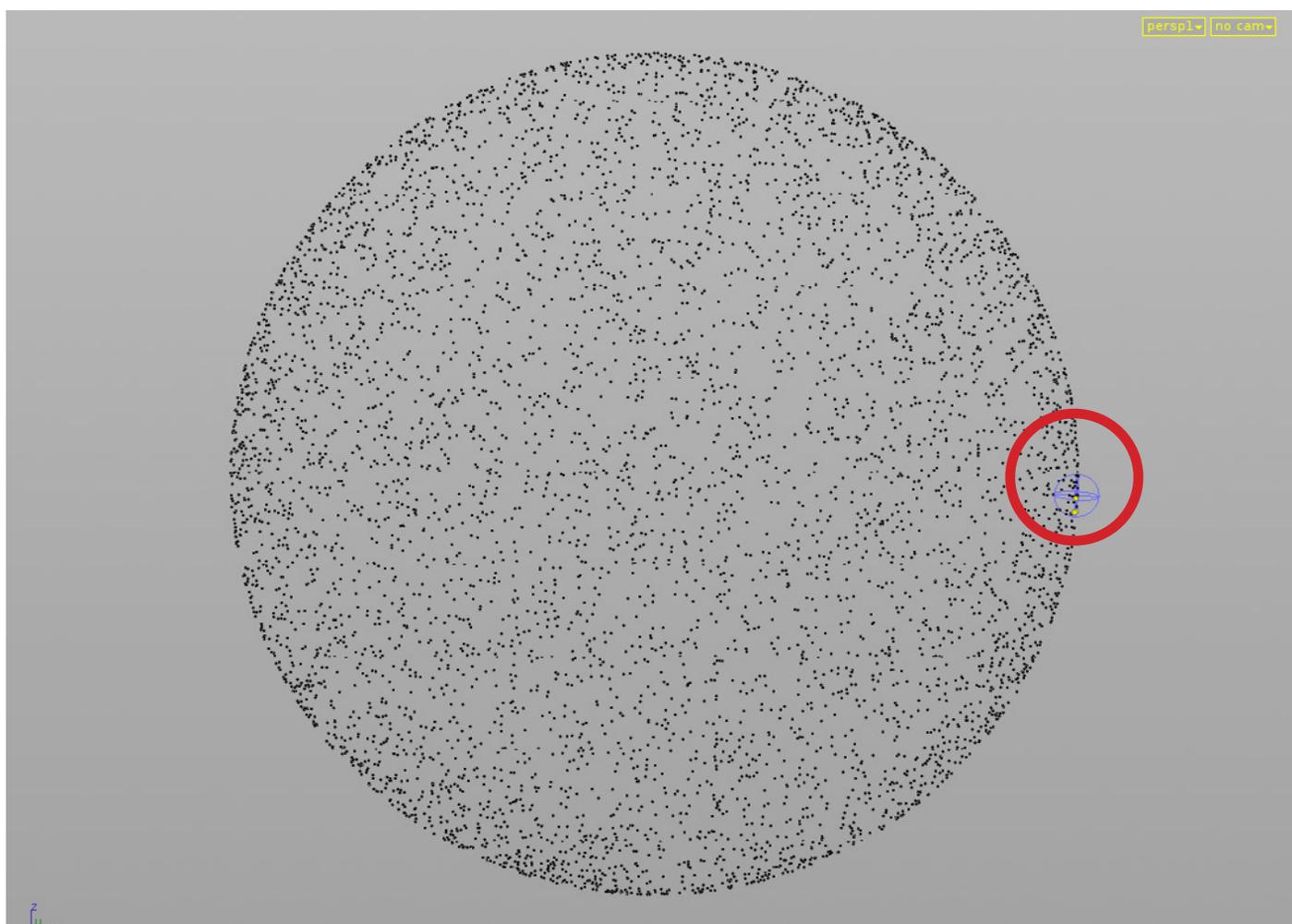
Plus bas, dans l'onglet *Number*, décocher le *Enable*, car nous n'avons pas besoin de cette option.



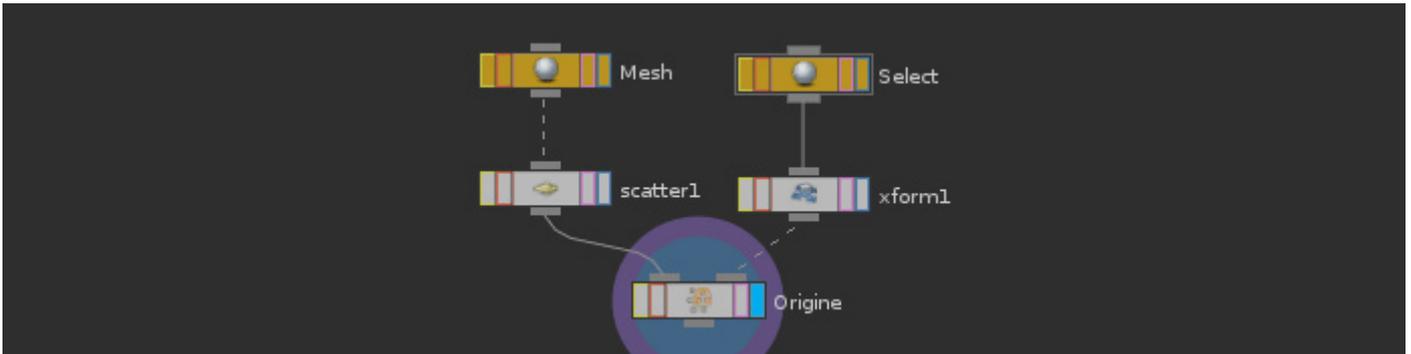
Dans l'onglet *Bounding*, cocher la case *Enable* afin d'activer la sélection via une géométrie. Puis mettre le *Bounding Type* sur : *Bounding Object*, car nous nous servons d'un mesh personnalisé pour sélectionner les points.



Ainsi on peut voir dans la Scene View que les points contenus dans la *sphere2* sont devenus jaunes, ils font partie à présent du groupe : **origine00**, et pourront être manipulés plus tard dans d'autres nodes indépendamment des autres points du scatter.



Renommer le node *group2* : **Origine**.
 Renommer le node *sphere2* : **Select**.
 Renommer le node *sphere1* : **Mesh**.



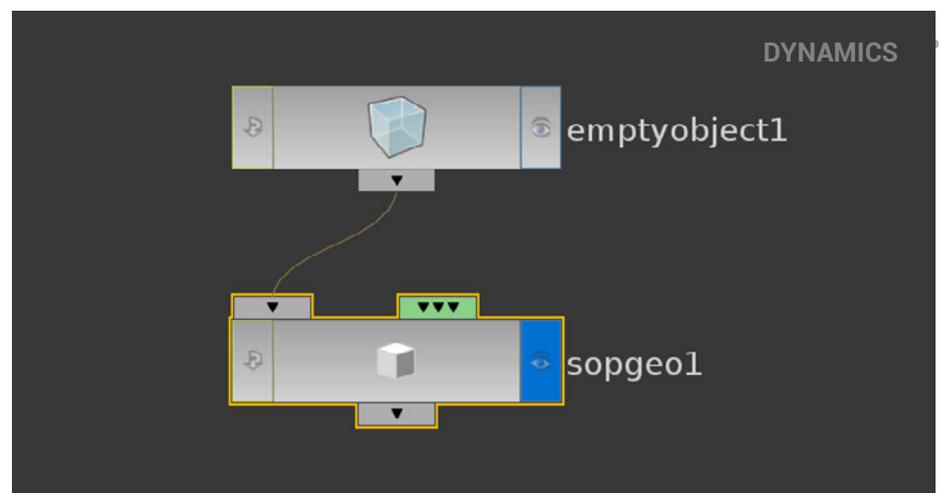
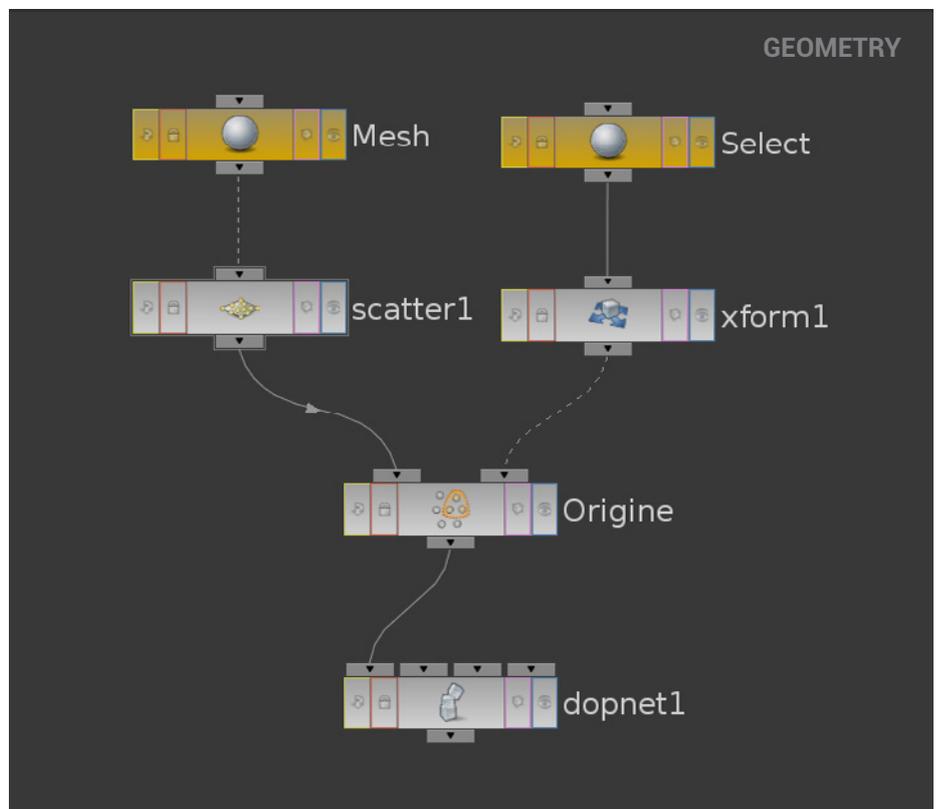
CRÉER L'EFFET DE DISPERSION :

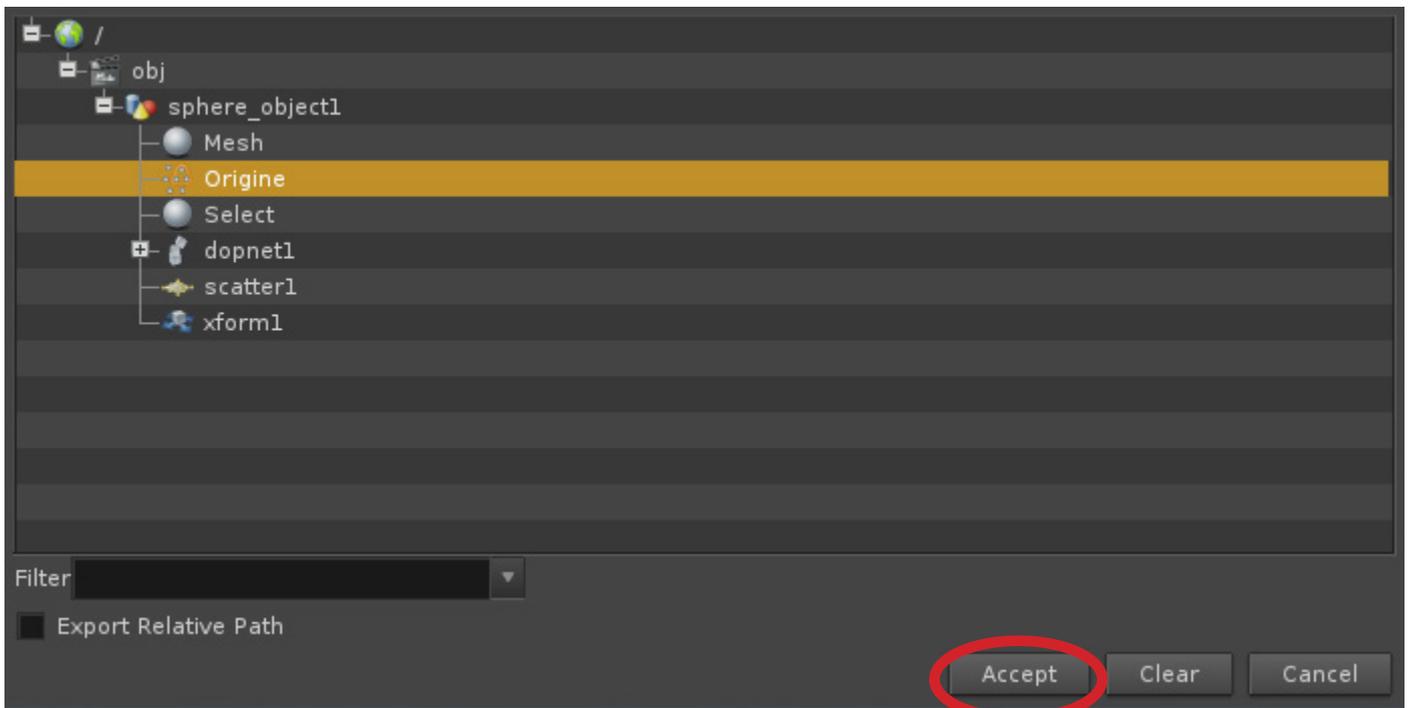
Brancher en dessous du node *Origine* un node *DOP Network* (premier Input).
 Il contient un espace de simulation (Dynamics - DOP) dans lequel nous allons pouvoir créer l'effet de dispersion.

Double-cliquer sur le node afin de rentrer à l'intérieur.
 Créer un node *Empty Object*.
 Puis relier son output au premier input d'un node *SOP Geometry*.

Ce node va aller chercher une géométrie qui se trouve dans l'espace Geometry (SOP) et la ramener dans l'espace de Simulation (DOP).

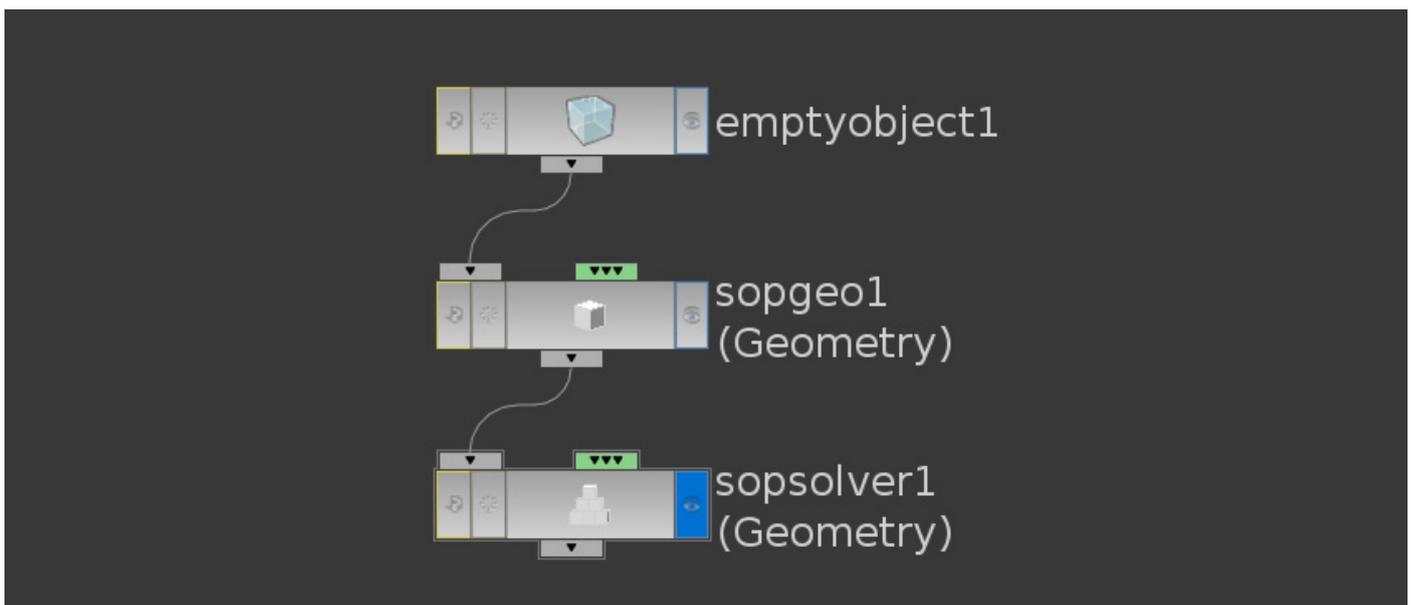
Dans le node *SOP Geometry* cliquer sur l'icône  placé à la droite de l'option *SOP Path*, une fenêtre s'ouvre, elle correspond à l'arborescence de notre projet, sélectionner *Origine*.





Ainsi les points sélectionnés par le groupe Origine sont importés dans l'espace de simulation.

Connecter au output du node *sopgeo1* un node *SOP Solver*.



Le *SOP Solver* va permettre au DOP d'utiliser un élément issu du SOP et de le faire évoluer au cours du temps. C'est dans son espace que nous allons mettre en place l'effet de dispersion.

Double-cliquer sur le node *sopsolver1* afin de rentrer dedans.

Relier à l'Output du node *dop_geometry* un node *Delete*.

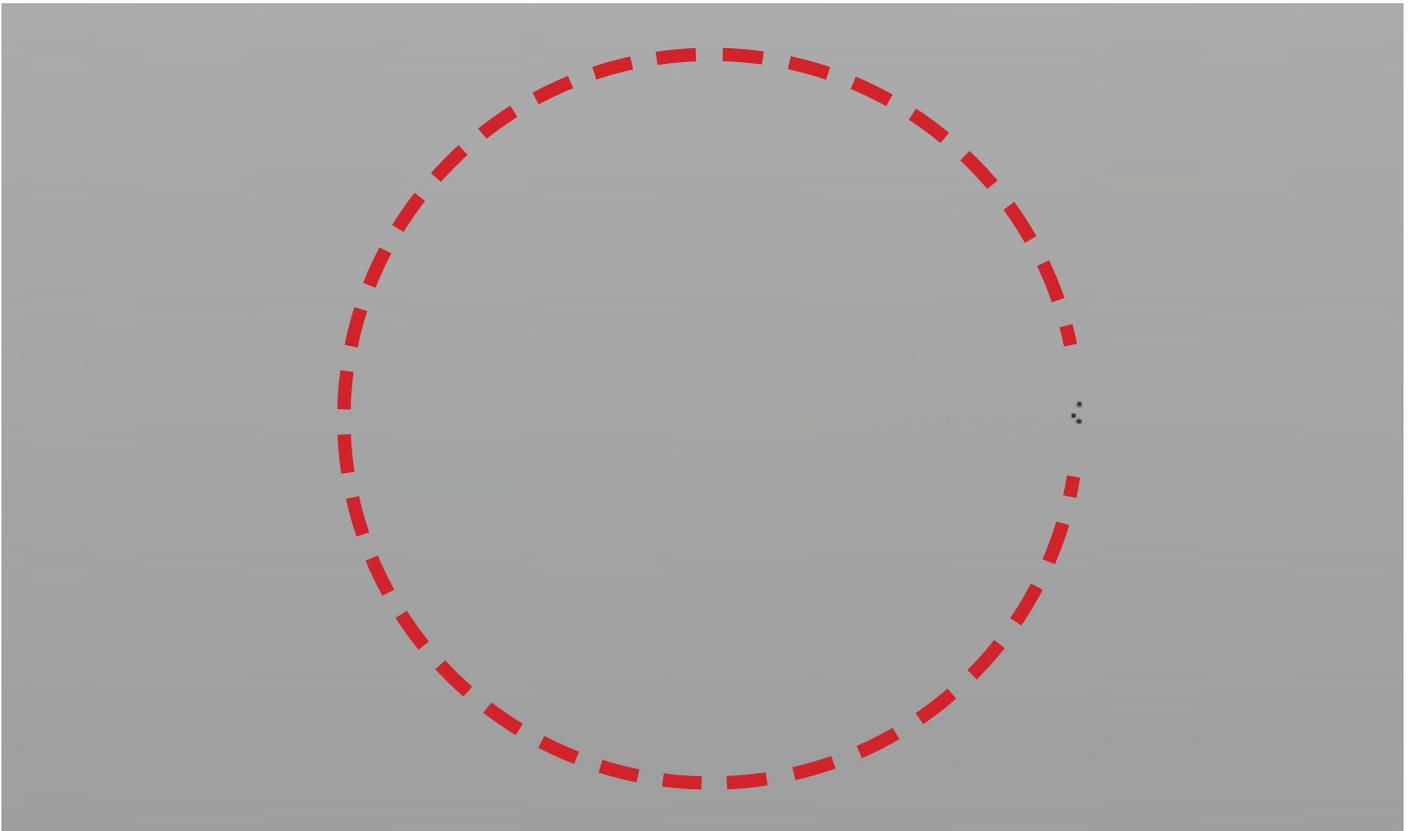
Charger dans le paramètre *Group* du node *Delete* : **origine00** en cliquant sur l'icône .



Mettre le paramètre *Operation* sur : **Delet Non_Selected**

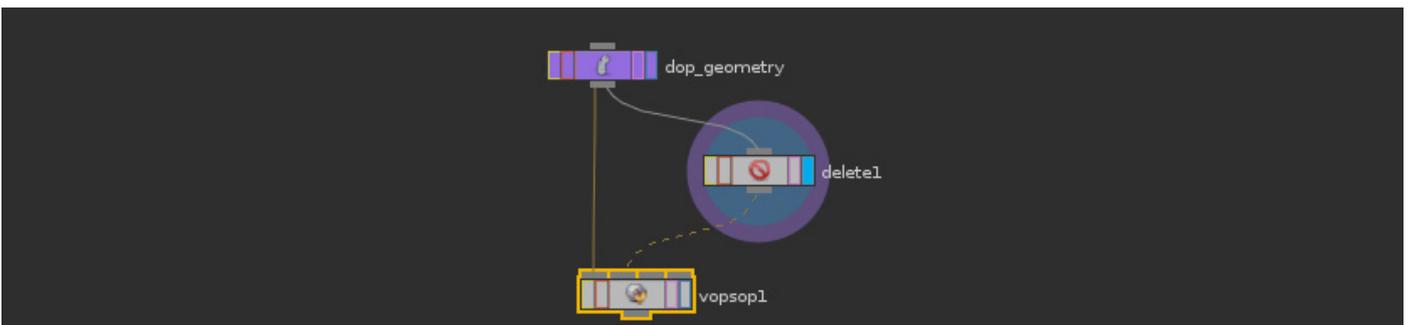
Puis *Entity* sur : **Point**

Ainsi on supprime dans le nuage de point du scatter, tous les points qui ne font pas partie du groupe *Origine00*. Il ne reste alors plus que les 3 points sélectionnés dans le groupe *Origine00*.



Les 3 points sont ici à droite, les pointillés rouges illustrent la zone originale de scatter.

Créer un node *VOP SOP*, et mettre dans son premier input le node *dop_geometry*, puis dans son second input le node *delete1*.



Le VOP SOP est un espace dans lequel on va pouvoir manipuler les points par le biais du langage VEX. Il permet de réaliser de petits "programmes" à l'intérieur d'Houdini.

Il peut être contrôlé par node ou directement par code, en voici un exemple :

```
void print(basis b) {  
    printf("basis: { i: %s, j: %s, k: %s }\n", b.i, b.j, b.k);  
}  
void print(matrix m) {  
    printf("matrix: %s\n", m);  
}
```

Rentrer dans le node *vopsop1* en double cliquant dessus.

Créer un node *Point Cloud Open*.

Ce node va chercher au sein d'un nuage de point (dans notre cas le scatter) tous les points situés autour d'un point d'origine (dans notre cas le groupe *origine00*).

Le paramètre *Point Cloud Texture* du *Point Cloud Open* sert à lui indiquer quel nuage de point utiliser.

Sélectionner le node *Point Cloud Open*.

Faire sortir le paramètre *Point Cloud Texture*, en cliquant sur l'icône suivant :



Puis sur *Promote Parameter*.

"Promoter" un paramètre permet de le rendre accessible en dehors du VEX Builder, autrement dit dans les paramètres du *vopsop*. Par conséquent, on pourra accéder au paramètre *Point Cloud Texture* dans les options du node *vopsop1*.

Faire sortir de la même manière le *Search Radius*, il nous servira à régler la zone de recherche qu'utiliserons chaque point pour répandre la couleur.

Sortir du **VEX Builder** afin de paramétrer le *Point cloud Texture*.

PUIS écrire dans le paramètre *Point cloud Texture* du *vopsop1* l'expression suivante :

```
op:`opinputpath(".",1)`
```

Note :

op : Le point cloud texture a habituellement besoin d'un fichier *.pc*, or nous souhaitons lui faire utiliser un nuage de point qui se trouve encore dans Houdini, et qui n'a donc pas été exporté en *.pc*.

op va permettre à Houdini, de lire un node à la place d'un fichier.

opinputpath() : permet de sélectionner dans la hiérarchie un node connecté à un input spécifique, ici nous souhaitons utiliser la géométrie qui se trouve dans le second input du node *vopsop1*.

Ce node à besoin de deux paramètres pour savoir où chercher : le nom/chemin du node et le numéro de l'input dans lequel est connecté la géométrie recherchée.

"." signifie : dans moi même.

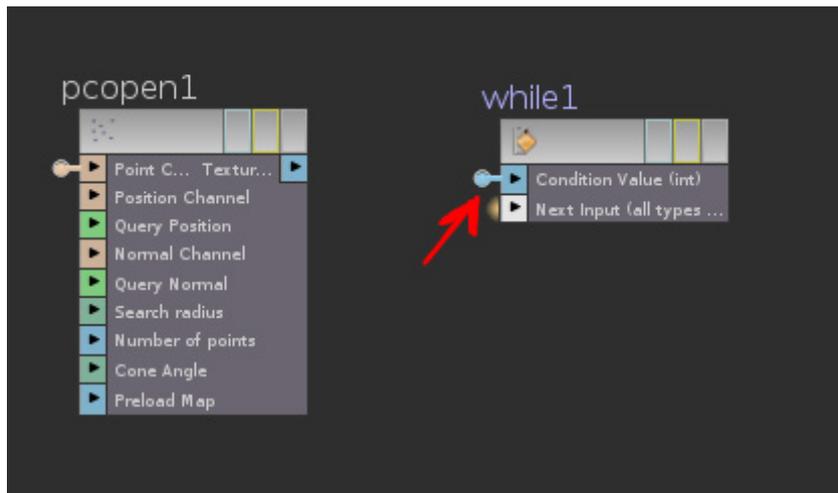
1 signifie : le second input (le premier étant nommé 0).

Les Backticks : `` que l'on retrouve avant et après le `opinputpath` sont à ne pas confondre avec des guillemets. Ils permettent d'exécuter une expression au sein d'une chaîne de caractère.

On peut donc conclure que l'expression va chercher ce qui se trouve dans le premier input du node *vopsop1*, c'est-à-dire les 3 points sélectionnés par le groupe *origine00* issus du node *delete1*.

Double cliquer de nouveau sur le node *vopsop1*.

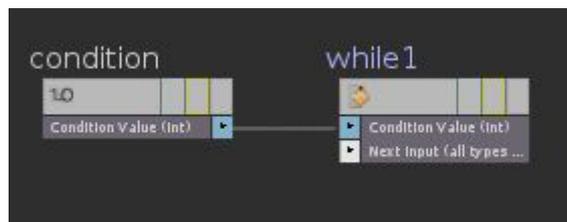
Créer un node *while loop*, puis faire un **Clic-Milieu** sur son paramètre *Condition Value*, Sélectionner l'option : **constant**



Ensuite, cliquer sur le node *Condition* relié à l'input *Condition Value*.
Puis mettre **1** en *Integer Default*.

Enfin, connecter le output du *point cloud* au *Next Input* du node *While* :

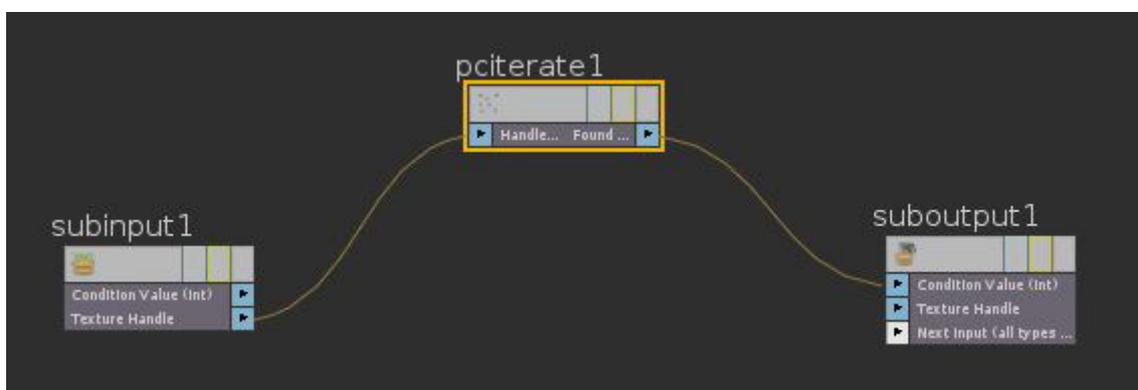
Mettre le *Condition* du *while1* sur **True**. (si Houdini se met à calculer sans s'arrêter, laisser le paramètre sur *False*, il suffira de remettre sur *True* une fois que les connexions à l'intérieur du node seront terminées.)



Rentrer dans le *while1*.

Créer un node *Point cloud iterate*.

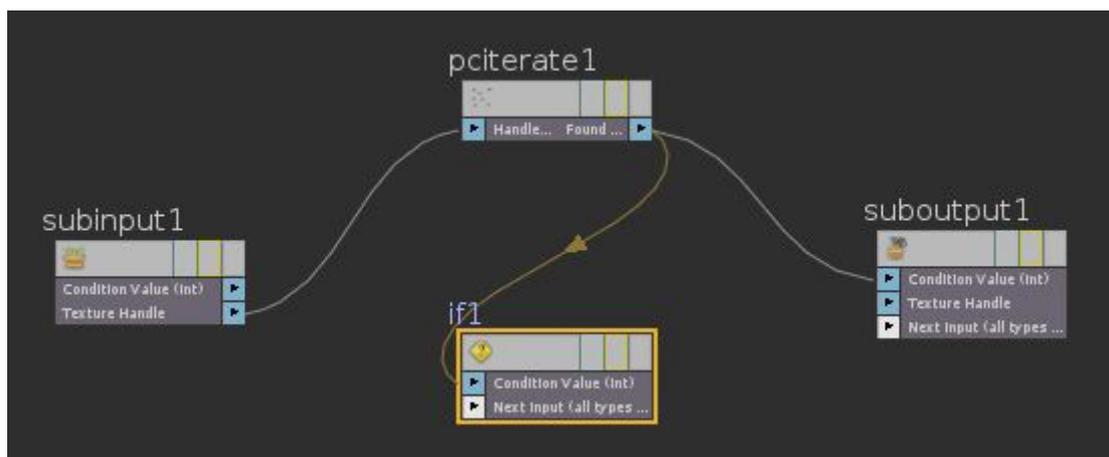
Brancher l'Output *Texture handle* du *subinput1* à l'Input *Handle* du *pciterate1*, puis on met le output de ce dernier dans le *Condition Value* du *suboutput1*.



Ainsi la boucle est en place.

Créer à présent un node *ifThenBlock*.

Brancher l'output du *pciterate* dans le input *Condition Value* du node *if1*, ainsi à chaque itération Houdini ira dans le node *if*.

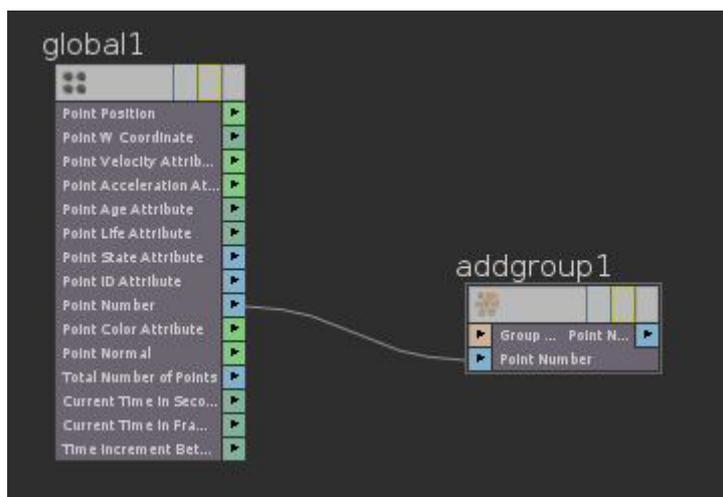


Rentrer dans le node *if1*.

Ajouter un node *Add point To Group*.

Écrire dans *Group* : **origine00**.

Puis créer un node *Global Variables*, relier son *Point Number* à celui du node *addgroup1*.



Note :

Le node *While* va exécuter en boucle les nodes qu'il contient tant que la condition est différente de 0.

À l'intérieur, le node *pciterate1* va permettre de faire avancer la boucle tant qu'il y aura des points à utiliser dans le *point cloud*. Quand une itération est effectuée il retourne 1 et enclenche le node *if1*.

Au sein du node *if1*, on ajoute au groupe *origine00* les numéros des points sélectionnés dans le *pciterate* à chaque itération.

Ainsi, au fur et à mesure des frames, le groupe *origine00* contient de plus en plus de points.

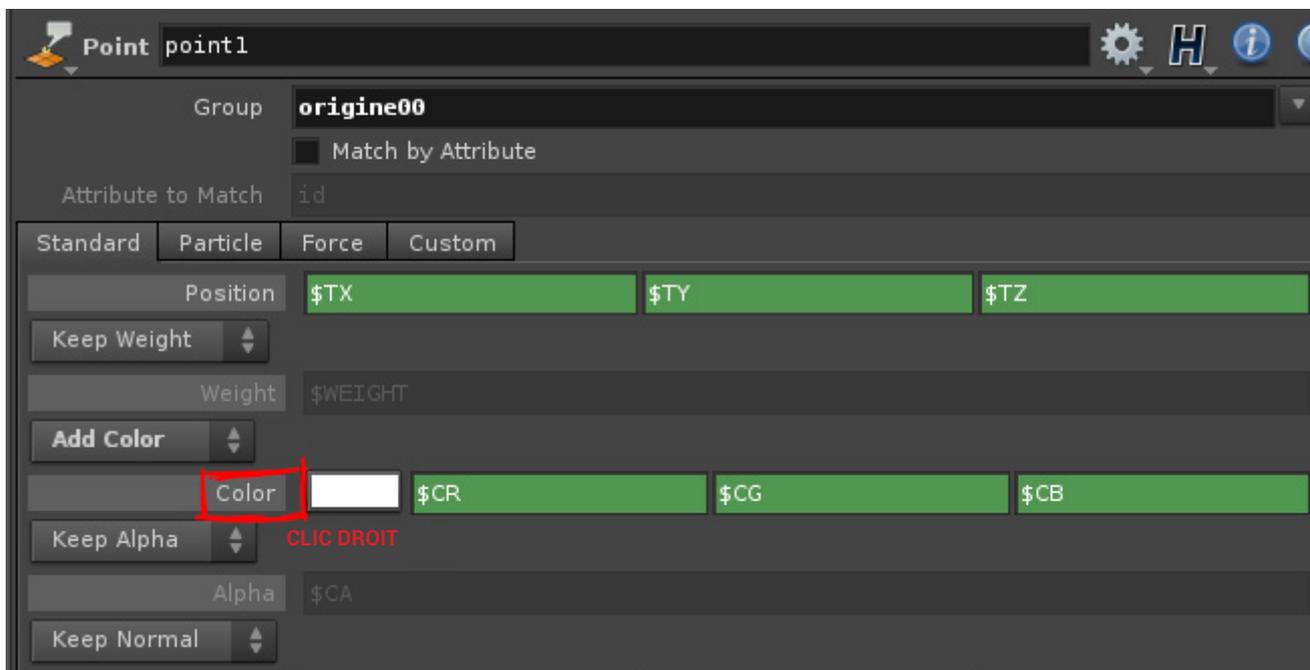
APPLIQUER LA COULEUR :

Retourner dans le node *sphere_object1* à la base du projet, et ajouter en dessous du node *dopnet1* un node *point*.

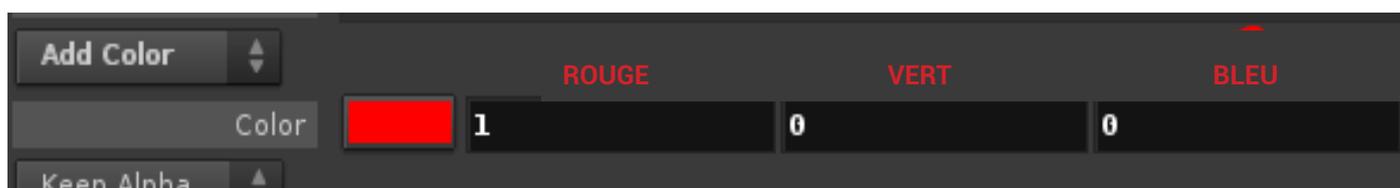
Mettre dans son paramètre *Group* : **origine00**

Puis dans l'onglet **Standard** cliquer sur le *Keep Color*, et sélectionner *Add Color*.

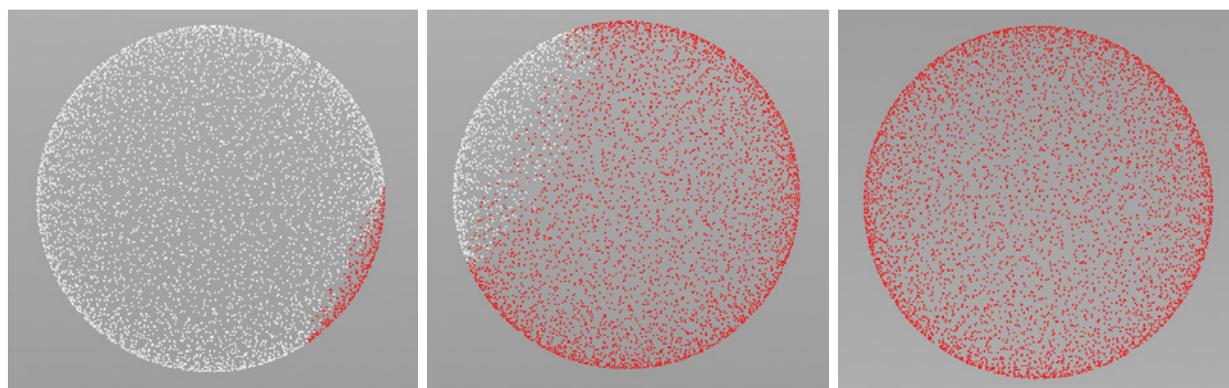
Effectuer un **Clic-Droit** sur *Color*, et sélectionner *Delet Channels*



Puis remplacer les valeurs du Rouge, du Vert et du Bleu comme suit : **1 0 0**, afin d'appliquer une couleur rouge aux points.



Si on lit l'animation, on peut voir que les points deviennent tour à tour progressivement rouges :

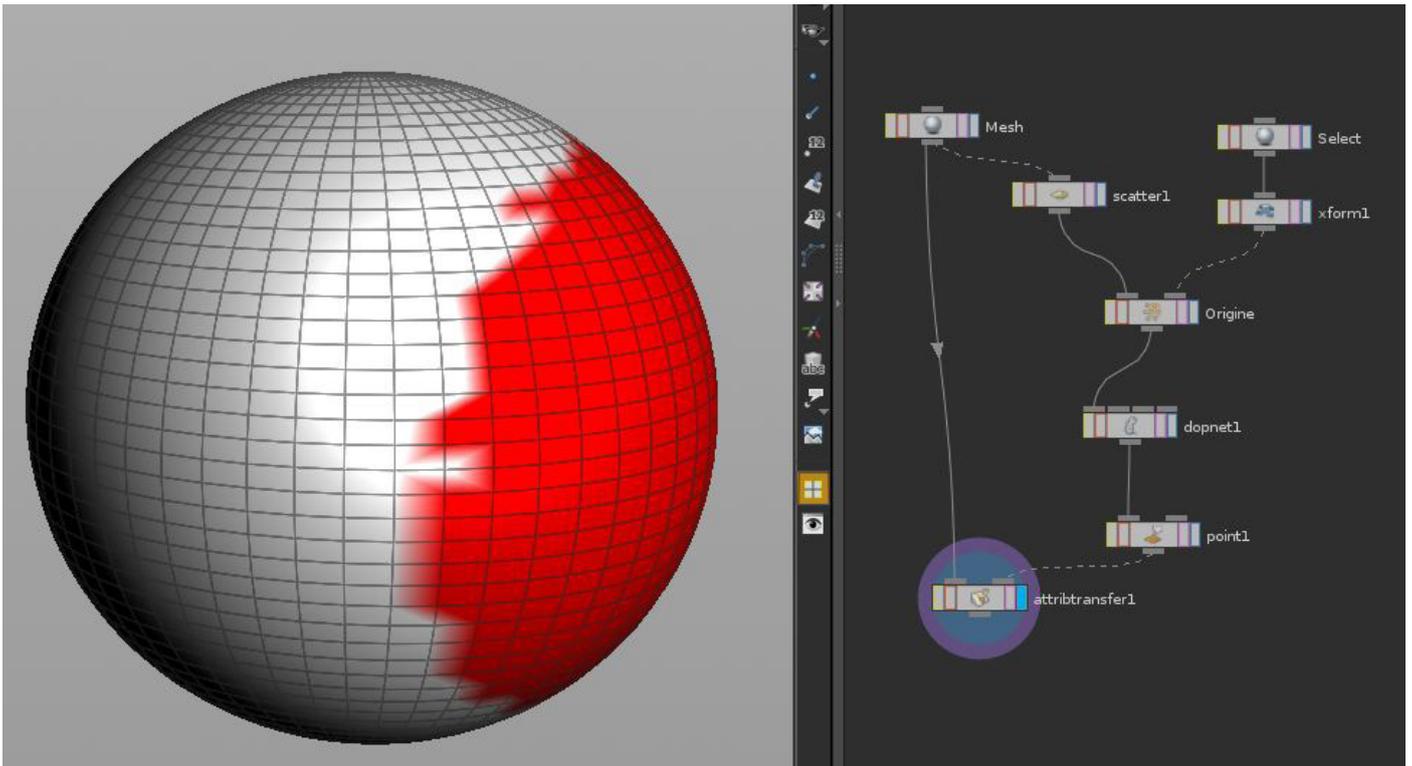


Nous allons transférer la couleur des points sur la sphère :

Créer un node *Attribut Transfer*.

Mettre dans son premier input le node *Mesh*.

Puis dans son second input le node *point1*.



Nous allons supprimer les polygones rouges :

Relier à l'output du node *attribtransfer1* un node *delete*.

Mettre le paramètre *Operation* sur :

Delet Non Selected.

Dans son onglet *Number* mettre le paramètre *Operation* sur : **Delet by Expression.**

Écrire dans *Filter Expression* : **$\$CR < 1$**

(Soit : la valeur du channel rouge est plus petite que 1)

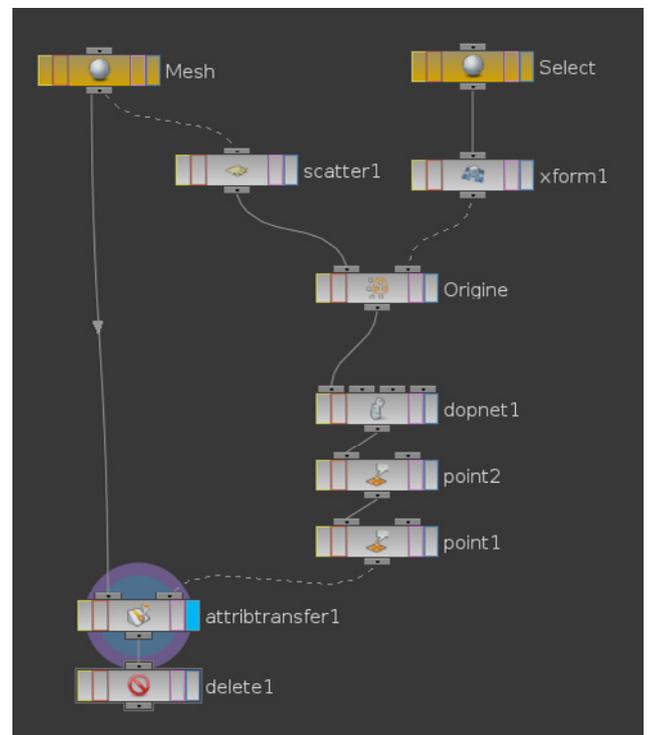
Puis ajouter un node *point* entre le node *dopnet1* et le node *point1*.

Dans l'onglet *Standard* régler le *Keep Color* sur *Add Color*.

Supprimer ses channels, puis régler la couleur sur :

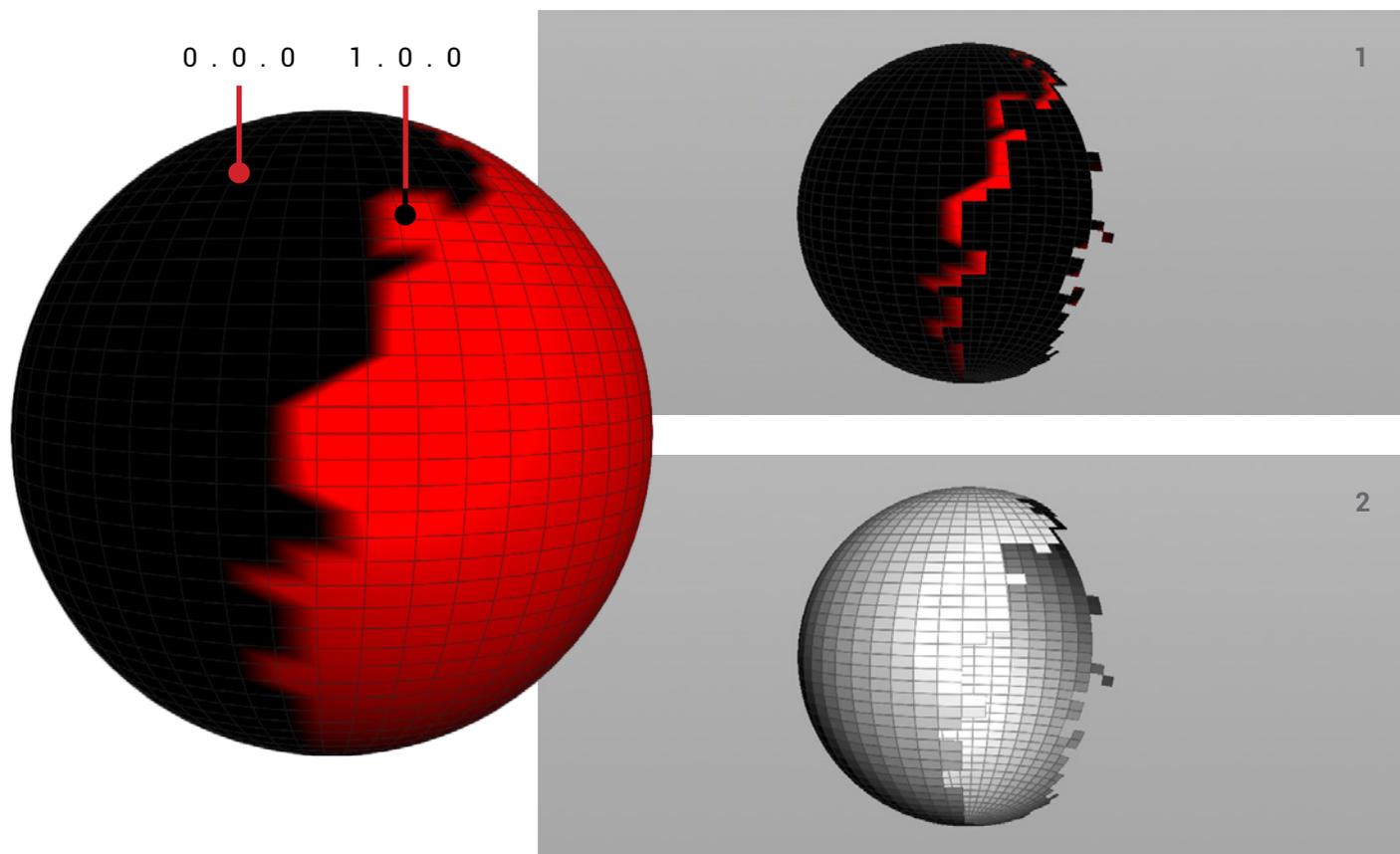
0 0 0

Ainsi nous avons d'une part des polygones qui ont des valeurs de rouge inférieures à 1 (noir), et d'autre part des polygones avec une valeur de rouge égale à 1 (rouge).



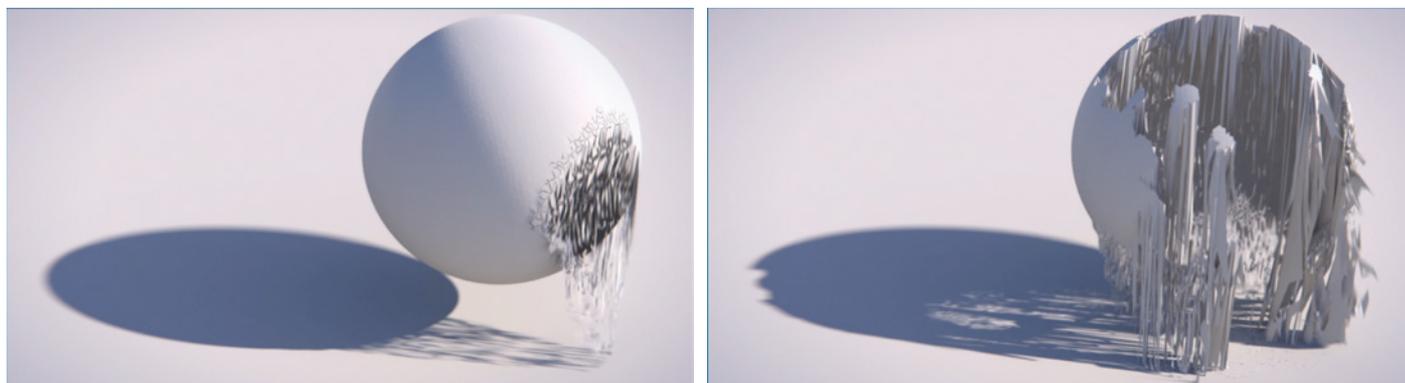
Si on active le node *delete* on peu constater qu'il supprime tous les polygones rouges¹.

Enfin effacer toute information de couleur sur l'objet, en créant au-dessous du node *Delete* un node *Attribut*. Dans l'onglet **Point** écrire au paramètre *Delete Attribut* : **Cd**
Il supprimera les informations de couleur stockées dans les points. L'objet redeviendra blanc².



L' effet n'est pas très impressionnant sur une sphère, mais si on applique ce principe à un mesh beaucoup plus complexe, un visage par exemple, que l'on multiplie les sources d'origine de la couleur, que l'on ajoute un noise procédurale sur la couleur afin de créer des variations, que l'on module la vitesse de progression, et que l'on varie la répartition des points du scatter, alors on peut avoir un résultat intéressant.

Ci-dessous un exemple avec une sphère à qui l'on applique un node *Spring* au niveau des points rouges :



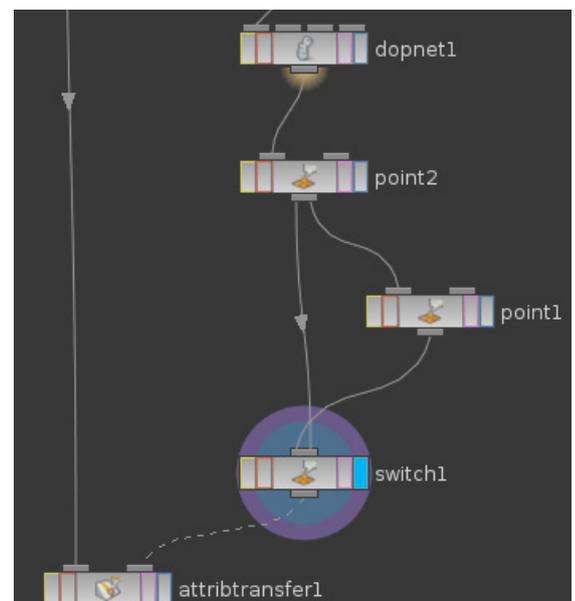
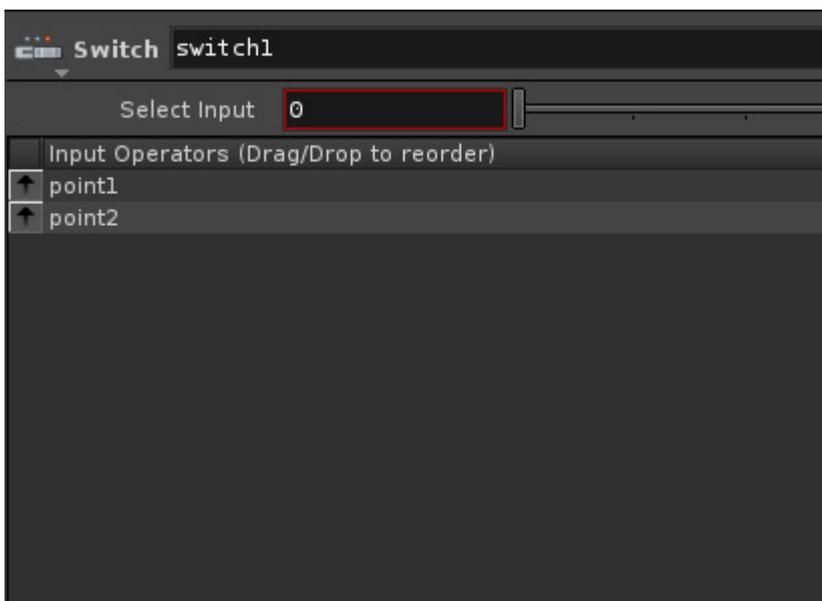
ASTUCE POUR ÉVITER D'AVOIR DES POLYGONES SUPPRIMÉS DÈS LA PREMIÈRE FRAME :

Créer un node *Switch* entre le node *point1* et le node *attribut transfer1*, puis relier également le output du node *point2* à l'Input du node *switch*.

Nous souhaitons n'avoir que des points noirs à la première frame, et faire apparaître les points rouges à la seconde. Le *switch* va permettre de lire en premier lieu soit l'output du node *point2* (noir) soit celui du node *point1* (rouge & noir). Il nous suffit alors de mettre des clefs dans son paramètre *Select Input* qui permet de passer de l'un à l'autre.

À la première frame mettre le *Select Input* à : **1**, puis placer une clef (**ALT + clic gauche**).

À la seconde frame mettre le *Select Input* à : **0**, puis placer une clef.

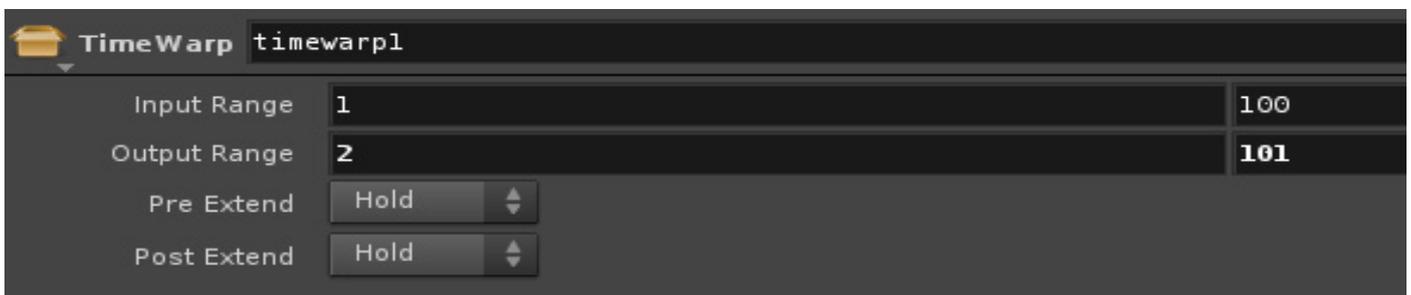


Nous allons ensuite décaler d'une frame l'animation issue du node *point1*, car ce dernier n'est visible qu'à la seconde frame, alors que son animation commence à la première.

Créer un node *TimeWrap* entre le node *point1* et le node *switch1*.

Il va permettre de décaler dans le temps l'animation de dispersion, en remappant les valeurs de l'*Input Range* à celle de l'*Output Range*.

Augmenter donc d'une unité les valeurs de l'*Output Range*.



L'effet devrait à présent fonctionner correctement.