Université Paris 8

Spécialité de Master *Arts et Technologies de l'Image Virtuelle*

de la mention *Arts Plastiques et Art Contemporain*

PYROTECHNIE DE SYNTHÈSE

Ivan J. MERSIC

Mémoire de Master 2

2012- 2013

EXTRAIT

PYROTECHNIE DE SYNTHESE

Ce mémoire traite des techniques permettant de simuler des effets pyrotechniques virtuels de manière réaliste. Il s'articule autour de trois axes ; se documenter — partie dans laquelle j'expose les différents outils et techniques dynamiques existantes afin de me demander en quoi elles pourraient me servir lors de la réalisation d'effets pyrotechniques —, tester — une partie consacrée à une phase d'expérimentations et de petits projets basiques autour de différentes techniques — et enfin, reproduire — une dernier axe tournant autour de la reproduction d'une célèbre scène de Matrix, le crash de l'hélicoptère.

ABSTRACT

SYNTHESIS PYROTECHNICS

This master thesis deals with technics used to simulate realistic computer-based pyrotechnic effects. We will discuss the technics during three main chapters ; gathering information — a section in which I will expose the different existing dynamic tools and technics in order to wonder how we could use this in our pyrotechnic effects —, testing — in this part we will go through several experiments and play with different technics by creating basic scenes and projects — and finally, replicate — in this last chapter I will try to recreate the exact same effects used on the famous scene of The Matrix, helicopter's crash.

Pour m’avoir aidé à mener à bien mon mémoire je voudrais adresser mes plus sincères remerciements à Marie-Hélène Tramus, Cédric Plessiet et Anne-Laure George-Molland qui n’ont pas hésité à me conseiller et me soutenir même lorsque je n’étais pas sûr de moi. Merci à Corentin, mon acolyte toujours prêt à rendre service. Merci à Reyes pour son amour sa tendresse et son soutien constant. Merci à Dytho, Arthur, Freddy, Chanel, Solène pour leurs conseils, leur humour et leur réconfort en cas de coup dur. Merci également à Luis Pages, sans qui mon mémoire aurait sans doute un peu moins de contenu. Enfin merci à mes parents et grands-parents pour ne jamais avoir douté de moi et pour toujours avoir été là lorsque j’en avais besoin.

SOMMAIRE

Introduction

La pyrotechnie, « *l'art et la science de créer du feu et des explosions*[1] » n'a pas attendu la naissance du cinéma en 1895 pour émerveiller les foules. En effet, tout commence en l'an 700, lorsque les trois puissances méditerranéennes — byzantins, grecs et romains — mettent au point un savant mélange de trois composés — salpêtre, soufre et charbon —, donnant naissance à une poussière magique connue sous le nom de poudre noire. Cette substance hautement inflammable, longtemps utilisée comme compagnon de guerre pour incendier les camps ennemis, a dû attendre le IX^{ème} siècle pour accéder, en Chine, au rang de festivité ; c'est la naissance du feu d'artifice. Cette pratique a su avec le temps s'imposer dans nos mœurs, si bien qu'aujourd'hui, il serait difficile d'imaginer une fête nationale sans ce spectacle détonant. Ainsi, il n'est pas étonnant de voir le cinéma — un autre art du spectacle — s'emparer de cet artifice pour donner à ses spectateurs une étincelle de rêve en plus. Déjà présent chez Méliès, les techniques pyrotechniques ont sues évoluer avec leur temps, mais ont tout de même conservé les mêmes procédés fondamentaux. Encore aujourd'hui, la poudre noire fait partie des matériaux principaux dans l'élaboration d'explosions à destination du Grand Écran. En témoigne l'une des techniques les plus utilisées dans ce domaine depuis de nombreuses années, le mortier. Cette méthode consiste à remplir une petite boite de poudre noire — que l'on appelle dans le métier, "bombe" —, et à la placer dans un long tube métallique — le mortier — avant d'y mettre le feu à l'aide d'un câble d'allumage. Au moment de la détonation, la bombe de poudre noire crée un souffle puissant, accompagné d'un flash lumineux et d'une épaisse fumée noire qui s'élève dans le ciel. Au cinéma, les effets pyrotechniques deviennent rapidement l'un des ingrédients indispensables aux films d'actions, et finissent par devenir la marque de fabrique d'un

1: Définition traduite du livre ***Special Effects the History and Technique*** (« *Pyrotechnics [are] the art and science of creating fire and explosions*»).

genre de production, le blockbuster. Dans les années 90, le septième art va connaitre un tournant important avec l'apparition des premiers longs-métrages intégrant des images générées par ordinateur. Petit à petit, ces techniques d'effets spéciaux vont se perfectionner, jusqu'à atteindre une qualité capable d'égaler des effets qui étaient jusqu'alors réalisés uniquement sur le tournage. Il se forme alors une question dans les esprits communs, les effets spéciaux numériques vont-ils faire disparaitre les techniques de plateaux ? Lorsque l'on pose cette question à Dave Crownshaw, superviseur des effets de neige sur des films comme ***Le Jour d'Après*** ou ***Quantum of Solace***, il nous révèle que, dans son cas, les techniques modernes ont produit l'effet inverse : « Les producteurs avaient l'habitude de supprimer de leurs scénarios les scènes contenant beaucoup de neige car ils savaient ce que cela impliquait. Aujourd'hui ce n'est pas un problème. Les gars de la 3D font un super boulot lorsqu'il s'agit de peindre de vastes paysages blancs, et nous pouvons produire un travail de meilleure qualité avec lequel les acteurs peuvent interagir dans des zones d'avant-plan plus petites. Nous faisons maintenant plus de neige que jamais auparavant, donc à vrai dire, les gens de la 3D on les aime ! [2]». Le domaine de la pyrotechnie est également concerné par ces avancées technologiques. Bien qu'il soit encore fréquent de réaliser des explosions directement sur le décor, au fil des années, les effets spéciaux numériques, y compris dans ce domaine, ont pris une place de plus en plus importante, et il n'est pas rare de voir certains long-métrages réalisés uniquement avec des explosions en images de synthèse. Si cette évolution a été possible c'est grâce à des techniques de plus en plus évoluées qui pourraient réussir à tromper même l'œil le plus averti. C'est cet aspect que je trouve tout particulièrement intéressant. Je me suis alors demandé comment une image de synthèse pouvait simuler de façon crédible un effet aussi complexe qu'une explosion.

2: Citation traduite du livre ***Special Effects the History and Technique*** (« *Producers used to delete really big snow scenes from their scripts because they knew what it could involve. Today, snow isn't a problem. The CGI guys can do a great job of painting vast distant landscapes white, and we can concentrate on doing better quality work that performers can interact with in the smaller foreground areas. We now do more snow than ever before, so we actually like the CGI people!* »).

Comment peut-on réaliser une simulation pyrotechnique virtuelle convaincante ? C'est autour de cette problématique que j'ai choisi d'articuler mon mémoire. Néanmoins, dans ma démarche de réalisme et mon processus de crédibilisation d'un effet pyrotechnique, je ne souhaite pas m'arrêter simplement à l'effet en lui-même. En effet, lors d'une explosion, par exemple, beaucoup d'éléments vont entrer en jeu qui vont contribuer à nous faire croire à l'effet, à commencer par l'interaction avec les éléments des alentours. Destruction de décors, déplacements d'objets, éjection de débris, autant de techniques qu'il est possible de réaliser en image de synthèses et sur lesquels je souhaiterais également me pencher. Ainsi, j'organiserai mes recherches en trois temps : se documenter, expérimenter et reproduire. Ma première partie sera donc un axe de recherche et d'apprentissage sur les différentes techniques nécessaires à la réalisation d'un effet pyrotechnique virtuel. Mon second axe s'orientera davantage vers la réalisation de tests au vue des différentes techniques apprises. Enfin, je terminerai par une étude de cas qui sera l'occasion de mettre toutes ces connaissances en pratique, en entreprenant la reproduction des effets de la célèbre scène du crash de l'hélicoptère dans ***Matrix***. En ce qui concerne le logiciel, j'ai choisi de travailler uniquement sur ***Autodesk Maya*** car j'y suis déjà relativement familier et que, compte tenu de l'ampleur du sujet de recherche, il serait impossible de rester focalisé sur les effets si je me retrouve submergé par les outils.

PARTIE 1

Des outils virtuels au service de la pyrotechnie

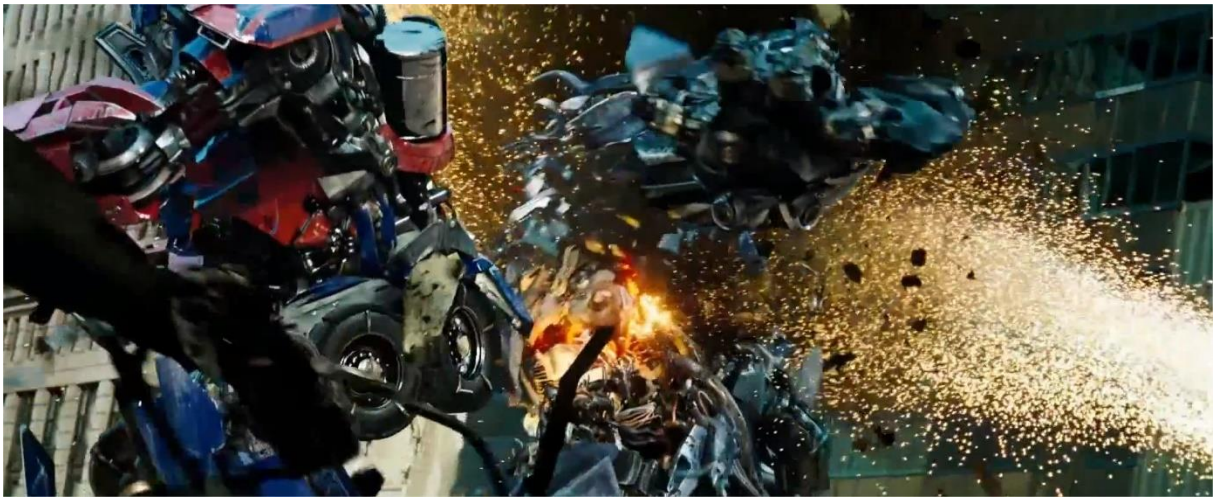
1.1. Les particules

Qu'est-ce que c'est ?

Le système de particules maya est un système dynamique qui consiste à faire réagir un nombre important de points dans un espace 3D en les affectant à l'aide de champs de force ou grâce à des outils de programmation, dans le but de simuler différents effets chaotiques, phénomènes naturels ou réactions chimiques. Bien que ce système fût largement utilisé il y a quelques années pour simuler des effets tels que le feu, les explosions ou la fumée, aujourd'hui les techniques ayant évoluées, de tels effets seront plutôt l'œuvre de simulations de Fluides, système sur lequel nous nous pencherons plus tard. Néanmoins, les particules sont encore indispensables pour la réalisation d'effets dynamiques et occuperont une place importante dans mes recherches sur la pyrotechnie de synthèse. A quoi pourraient alors me servir les particules dans la mise en place d'effets pyrotechniques ?

À quoi ça sert ?

Première association évidente, les étincelles. En effet, une étincelle n'est autre qu'une particule incandescente émise à une très grande vitesse qui dure une fraction de secondes. Il nous serait donc très aisé d'utiliser le système de particules de ***Maya*** pour créer un tel phénomène. Une étincelle peut être produite de différentes façons. Par une réaction chimique par exemple, phénomène que l'on observe avec certains pétards ou bougies d'anniversaire. On peut également en obtenir en frottant deux métaux l'un contre l'autre, en allumant un feu de bois, ou même lors d'impacts de balles sur des surfaces métalliques.



Transformers 3 : La Face cachée de la Lune, Michael Bay, 2011.

Un autre effet qui correspond parfaitement aux spécificités du système de particules, le feu d'artifice. Un jet de plusieurs fusées qui émettent consécutivement une série de particules lumineuses colorées. Bien qu'il y ait de nombreuses variantes au niveau des formes, de la vitesse, des couleurs ou du nombre de particules, l'effet reste toujours fondamentalement le même. Il est également possible de voir certains feux d'artifices dont la première explosion génère un groupe de particules qui vont émettre à leur tour une autre série de particules quelques secondes plus tard.

Harry Potter et l'Ordre du Phœnix, David Yates, 2007.

Il faut sauver le soldat Ryan, Steven Spielberg, 1998.

Par ailleurs, les particules peuvent aussi servir à rajouter du détail dans certains effets. Elles peuvent par exemple être utilisées pour faire de la cendre lors d'une irruption volcanique voire même des jets de lave dans un plan large. Dans le cas d'une explosion, les particules peuvent nous permettre de créer des effets de fins éclats de verre, de bois, de sable, de terre, etc. Nous verrons qu'il est également possible de s'en servir pour faire des débris plus gros mais la technique utilisée possède des limitations qui nous pousseront d'avantage à utiliser des *Rigid Bodies*, sujet dont nous parleront évidemment aussi plus loin.

Enfin, le système de particules de *Maya* peut aussi être utilisé non pas comme outil principal mais comme outil supplémentaire dans l'utilisation d'autres systèmes. Je pense principalement aux Fluides que l'on va pouvoir émettre depuis des particules. Il va donc nous être possible d'utiliser une simulation de particules pour générer du feu, de la fumée ou encore des explosions. Elles peuvent alors soit servir d'émetteur principal soit permettre de rajouter du détails dans certaines simulations en venant sculpter l'effet d'origine — pour des projections de fumée ou de boules de feu lors d'une explosion par exemple.

Démineurs, Kathryn Bigelow, 2009.

Comment ça marche ?

Intéressons-nous à présent à la technique en elle-même. Tout d'abord, afin d'émettre nos particules dans Maya nous avons plusieurs possibilités en fonction de l'effet que nous souhaitons réaliser. S'il s'agit d'une bougie magique ou d'un feu d'artifice nous allons plutôt utiliser un émetteur de type omnidirectionnel (*Omni*) ou directionnel (*Directional*). Le premier permet d'émettre des particules dans toutes les directions tandis qu'avec le second nous allons pouvoir le faire dans un cône en réglant le paramètre *Spread* qui correspond à l'angle de diffusion des particules. L'émission se fera alors depuis un point dans l'espace. Si nous souhaitons néanmoins émettre depuis une plus large surface — pour créer par exemple une façade en feu ou une onde de poussière —, nous pouvons le faire grâce à un émetteur de type *volume* ou en émettant depuis un objet en choisissant l'option *Emit From Object*. Nous verrons également plus tard qu'il est possible d'émettre depuis les points d'une géométrie si nécessaire.

Afin de paramétrer la dynamique de nos particules nous avons accès à de nombreux attributs que nous allons pouvoir modifier manuellement ou à l'aide d'expressions ou de dégradés en fonction d'un autre paramètre. Pour gérer leur durée de vie, par exemple, nous pouvons modifier manuellement le paramètre *Lifespan* ou créer une expression ou un dégradé sur l'attribut *LifespanPP*. En effet, certains attributs possèdent une version qui contrôle la globalité des particules et une autre qui permet de les diriger individuellement — son nom possède alors le suffixe "PP" signifiant "par particule". Certains attributs ne possèdent qu'une version "par particule", ils n'ont donc pas de suffixe "PP", c'est le cas, par exemple, de l'attribut *Velocity* qui contrôle la vitesse.

Lorsque l'on travaille avec les particules *Maya*, il faut distinguer deux choses : l'émission des particules et leur comportement. Tandis que l'émetteur possède des paramètres concernant le comportement des particules au moment de leur création — vitesse initiale, direction d'émission, nombre de particules générées, etc. —, les particules, elles, vont avoir des paramètres concernant leur comportement au cours du temps — durée de vie, conservation de la vitesse, position, etc. — et les caractéristiques qui leur sont propres — masse, couleur, opacité, mode d'affichage, etc. Ainsi, si je souhaite que mes particules soient émises à une grande vitesse, je vais devoir augmenter l'attribut *Speed* de mon émetteur. Et si je veux que mes particules perdent rapidement leur vitesse, je vais baisser l'attribut *Conserve* des particules elles-mêmes. Dans le cas d'étincelles par exemple, la vitesse initiale (*Speed*) sera très élevée, la durée de vie (*Lifespan*) très courte et la conservation de la vitesse (*Conserve*) à son maximum.

Pour gérer l'affichage des particules nous avons accès à un paramètre de rendu qui nous permet de choisir entre différents types de particules. Le type de rendu par défaut, en *Points*, peut permettre de simuler déjà pas mal de choses comme des éclats d'eau, de sable ou de verre, par exemple. Pour un rendu d'étincelles, cependant, on préférera choisir le mode *Streak* qui permet de remplacer les particules par des lignes droites qui s'orientent et grandissent en fonction de leur vitesse. Il est également possible de régler d'autres paramètres de rendu tel que la couleur ou l'opacité. Si l'on souhaite, par exemple, faire scintiller les particules d'un feu d'artifice, nous avons la possibilité de connecter une texture de *Noise* à notre paramètre *OpacityPP* en fonction de la durée de vie de chaque particule (*LifespanPP*). Nous en reparlerons plus précisément lorsque nous passeront à la pratique.

1.2. Les instances

Qu’est-ce que c’est ?

Pour aller plus loin dans le rendu de nos particules, il existe une technique consistant à instancier une géométrie afin que l’affichage de chaque particule ne dépende plus du type de rendu choisi mais d’un objet de notre scène. Autrement dit, chaque particule de notre système va pouvoir être remplacée par la géométrie de notre choix. L’utilisation de cette technique peut ainsi être bénéfique lorsque l’on souhaite simuler un grand nombre d’objets dynamiques, car les temps de calculs seront beaucoup plus réduits que s’il fallait calculer les collisions de chaque objet indépendamment.

À quoi ça sert ?

Dans le cas d'effets pyrotechniques, instancier des particules peut nous permettre de créer des débris de pierre, de bois ou même de gros morceaux de verre projetés suite à une explosion. Cette méthode possède néanmoins des limitations. En effet, étant donné que les instances sont des géométries contrôlées par de simples points dans l'espace, si les particules entrent en collision avec une surface, les instances et la surface s'interpénètreront et la collision ne s'effectuera pas de manière précise. On aura ainsi intérêt à utiliser cette technique soit pour des plans larges, soit dans des plans où les débris n'entrent en collision avec aucun objet.

Inglourious Basterds, Quentin Tarentino, 2009.

Comment ça marche ?

Afin d'instancier un objet à des particules, il faut d'abord s'assurer de placer la géométrie au centre du monde et de réinitialiser ses transformations. Il faut également veiller à positionner correctement son point de pivot car c'est en ce point que l'objet va venir superposer la particule. Une fois les instances créées, chaque particule va être remplacée par une copie de l'objet en question mais restera néanmoins lié à lui. En effet, il est possible de modifier l'objet instancié, toutes les instances associées aux particules seront alors affectées. Nous pouvons également instancier un objet animé, le seul problème est que toutes les instances auront exactement la même animation. Il faudra donc créer autant d'objets animés que d'animations différentes souhaitées pour nos instances. La tâche pouvant être très fastidieuse, il existe des outils externes permettant de l'automatiser, comme le script *Instance Copier*. Néanmoins, dans notre cas, nous n'aurons pas à nous occuper de ce problème car nous utiliseront les instances principalement pour des débris qui n'auront pas besoin d'être animés au préalable. Quoi qu'il en soit, une fois nos instances créées, elles se ressemblent toutes. Afin de donner un rendu plus aléatoire à tous nos débris, nous avons à notre disposition une série de paramètres liés aux instances, auxquels nous allons pouvoir affecter n'importe quel attribut de nos particules, y compris des attributs que nous avons nous-même créés. Cela peut nous permettre, par exemple, de donner une taille aléatoire à chaque instance ou encore de les faire tourner en fonction de leur vitesse et donner ainsi un rendu plus chaotique à nos débris.

1.3. Les Goals

Qu’est-ce que c’est ?

Afin de contrôler nos particules, il existe également une technique appelée *Goal* qui consiste à définir un objet ***Maya*** qui va servir à attirer les particules plus ou moins rapidement en fonction de la puissance d’attraction paramétrée.

À quoi ça sert ?

Cet outil peut avoir différentes applications, la plus courante étant de venir donner une forme à un groupe de particules. Néanmoins, pour la réalisation d'effets pyrotechniques, l'utilisation de *Goals* sera à priori réduite à des effets secondaires, notamment pour des déformations de surfaces réalisées à l'aide de *Soft Bodies*, sujet dont nous parlerons plus tard. Nous pouvons aussi imaginer se servir de cette technique pour contrôler le mouvement d'un groupe d'oiseaux ou de poissons qui se disperseraient suite à une explosion par exemple.

2012, Roland Emmerich, 2009.

Comment ça marche ?

Un *Goal* peut être un polygone, un *NURBS*, une autre particule ou simplement le *Transform Node* de n'importe quel objet. Dans le cas d'une géométrie, toute animation appliquée à la forme de l'objet — qu'il soit déformé par un squelette, un *Cluster*, un *Lattice* ou des particules dans le cas d'un *Soft Body* — seront prise en compte par les particules lors de leur attraction.

Afin de contrôler la puissance de ce *Goal*, nous avons accès à un paramètre appelé le *Goal Weight*. Il s'agit de la vitesse à laquelle les particules vont être attirées par leur *Goal*. Il est également possible de paramétrer cette puissance d'attraction indépendamment pour chaque particule, à l'aide de l'attribut *Goal PP*. Lorsque l'on affecte une valeur à cet attribut, il est multiplié par la valeur du *Goal Weight* avant d'être affecté aux particules. Il est donc important, pour pouvoir correctement modifier le *Goal* de chaque particule, de mettre ce dernier à 1. La valeur du *Goal* sera alors précisément celle du *Goal PP*.

Les particules possédant un *Goal* peuvent toujours être affectées par des champs de force à condition que la valeur du *Goal Weight* ou *Goal PP* soit inférieur à 1. Dans le cas contraire, les particules seraient simplement contraintes au *Goal* sans avoir aucune possibilité de mouvement propre.

1.4. Les Soft Bodies

Qu'est-ce que c'est ?

Un *Soft Body* est une géométrie pour laquelle chaque vertex est contrôlé par une particule. Lorsque l'on crée un *Soft Body* à partir d'un objet, il est possible d'utiliser l'objet d'origine comme *Goal* pour le *Soft Body*, c'est-à-dire que le *Soft Body* tentera toujours de revenir à cette forme initiale.

À quoi ça sert ?

L'utilisation de cette technique, dans le cadre de mes recherches sur la pyrotechnie de synthèse, aura essentiellement à voir avec des déformations de surfaces. Cela peut être une surface de lave, une étendue d'eau affectée par le souffle d'une explosion, une plage de sable déformée par des débris, ou tout autre type de surface déformable allant du métal au simple tissu. Cette technique pourrait également être utilisée directement pour la déformation de gros débris. Néanmoins, il existe dans *Maya* d'autres systèmes plus performant tel que les *nCloth* ou le plug-in *DMM* qui nous permettront de réaliser ce genre d'effets de manière plus précise et plus réaliste. L'avantage, cependant, de l'utilisation d'un *Soft Body*, est qu'il va nous permettre d'utiliser toutes nos connaissances concernant les particules et les *Goals* — champs de force, *Goal PP*, expressions, etc. — et d'utiliser des outils qui ne seraient pas forcément disponible avec d'autres systèmes similaires. Une chose à noter cependant est l'absence d'auto-collisions contrairement aux autres systèmes plus poussés. Il est possible néanmoins de contourner ce problème en affectant à chaque particule un champ de force pour qu'elles se repoussent entre elles. Le résultat est cependant moins précis et ne marche que dans certains cas.

Matrix, Lana Wachowski, 1999.

Une autre utilisation qui me semble pertinente par rapport à ma problématique est la création d'un effet de bougie. En effet, la flamme d'une bougie étant souvent assez statique, il nous est possible de la modéliser au repos puis de la convertir en *Soft Body* simplement afin de venir la perturber légèrement. Il s'agirait ensuite de lui appliquer un *Shader* avec les bonnes couleurs et transparences. Nous nous intéresseront d'ailleurs précisément à cet effet lorsque nous passeront à la pratique.

Barry Lyndon, Stanley Kubrick, 1975.

Comment ça marche ?

Par défaut, lorsque l'on crée un *Soft Body* à partir d'une géométrie, ***Maya*** va créer automatiquement un système de particules qui va venir contrôler chaque vertex de cette géométrie. Ainsi, si l'on vient perturber les particules, la géométrie sera déformée à l'identique. On peut imaginer, par exemple, créer un *Soft Body* à partir d'un plan et venir affecter ses particules à l'aide d'une turbulence animée pour simuler une nappe de lave agitée.

Il est également possible de modifier les paramètres de création du *Soft Body* afin que les particules viennent contrôler non pas l'objet d'origine mais une copie de celui-ci, et que la géométrie originale soit utilisée comme *Goal* pour les particules. Ainsi, à chaque fois que l'objet d'origine sera animé ou déformé, le *Soft Body* tentera de le suivre avec un décalage plus ou moins grand, en fonction de la valeur du *Goal Weight*. À l'inverse, lorsque l'on va venir perturber les particules, le *Soft Body* va se déformer puis revenir à sa forme initiale, toujours en fonction de son attribut de *Goal Weight*. Cet attribut va ainsi jouer un rôle majeur dans la mise en place du type de matériaux de notre surface, car il va contrôler la vitesse à laquelle la surface retrouve sa forme initiale. Une surface élastique, par exemple, retrouvera sa forme d'origine rapidement tandis qu'une étendue d'eau mettra un certain temps. Dans le cas de la déformation d'une plage de sable, après avoir été perturbée, la surface ne reprendra jamais sa forme initiale, l'utilisation d'un *Goal* serait alors inutile.

Le matériau d'une surface ne dépend pas seulement de la vitesse à laquelle le *Soft Body* va reprendre sa forme initiale, elle dépend aussi de sa capacité à conserver ou perdre de l'énergie. En effet, lorsqu'une particule tente d'atteindre un *Goal*, si elle n'a aucune perte de vitesse, elle va venir graviter autour de ce *Goal* jusqu'à atteindre un équilibre et se stabiliser. Néanmoins, si l'on introduit la notion de perte d'énergie, la particule mettra toujours le même temps pour atteindre le *Goal*, cependant, étant

donné que sa vitesse diminuera au cours du temps, elle ne fera pas de va-et-vient autour du *Goal* et son parcours sera plus linéaire. Ainsi, la surface d'un *Soft Body* ne sera pas seulement caractérisée par son paramètre *Goal Weight* mais aussi par le paramètre de *Conserve* des particules qui lui sont attachées. Si nous laissons ce paramètre à 1, les particules du *Soft Body* ne perdront pas d'énergie et la surface effectuera un mouvement de haut en bas avant de se stabiliser, ce qui pourrait caractériser la surface d'un liquide par exemple. Par contre, si nous baissons suffisamment ce paramètre, le *Soft Body* reviendra directement à sa forme initiale, ce qui donnera à notre surface un aspect plus proche du caoutchouc ou de la guimauve — bien que je ne vois pas tellement ce que viendrais faire une guimauve dans un effet pyrotechnique.

Aussi, comme lors de l'utilisation d'un *Goal* sur de simples particules, nous avons la possibilité d'affecter un *Goal Weight* différent pour chaque particule et donc chaque vertex de notre *Soft Body*, grâce au paramètre *Goal PP*. Néanmoins, nous disposons d'un outil supplémentaire et beaucoup plus visuel pour leur affecter une valeur, l'*Attribute Paint Tool*. En effet, cet outil va nous permettre de peindre littéralement les valeurs de *Goal PP* sur chaque particule de notre *Soft Body*. On va ainsi pouvoir créer des surfaces possédant différentes caractéristiques et ainsi regrouper plusieurs matériaux au sein d'une même géométrie. Dans le cas de notre plage de sable, par exemple, nous pourrions imaginer que la partie de la plage loin de la mer serait constituée de sable sec et donc ne retrouverait jamais sa forme initiale en cas de déformation (*Goal PP* à 0), et la partie en contact avec la mer serait une surface imbibée d'eau qui mettrait un certain temps mais finirait toujours par redevenir plate (*Goal PP* supérieur à 0). Ainsi, avec un dégradé du *Goal PP* de 0 à 0,5 par exemple, nous pourrions créer une surface qui se déformera correctement quel que soit l'endroit où tombent les débris, et ce sans avoir à créer plusieurs *Soft Bodies*.

Enfin, lorsque l'on veut créer une surface qui réagit comme une étendue d'eau, régler le *Goal Weight* ne suffit pas. En effet, si nous lançons un objet dans de l'eau, celui-

ci va créer une vague qui va se propager sur toute la surface. Or, par défaut, si j'affecte une partie de mon *Soft Body*, seule cette partie sera déformée avant de revenir en position. Ainsi, pour pouvoir simuler cet effet de propagation, il est possible d'appliquer à nos particules une contrainte dynamique appelée *Springs* qui va venir lier chaque particule à l'aide de sortes de ressorts virtuels. De cette façon, le mouvement de chaque particule entraînera celui de ces voisines, recréant ainsi ce phénomène de propagation de la vague. À l'instar des paramètres de *Goal Weight* et de *Conserve*, deux nouveaux attributs vont alors être disponibles, pour paramétrer la dynamique de notre surface. L'attribut *Stifness* pour régler la vitesse à laquelle le ressort va revenir à sa taille de repos. Et l'attribut *Damping* qui correspond à la vitesse à laquelle le ressort perd de l'énergie. Notons, cependant, que si nous augmentons la valeur de ces attributs de manière trop importante, il est possible que le *Soft Body* explose — plutôt ironique quand on connaît mon sujet. Ainsi, pour pouvoir pousser les paramètres sans être confronté à ce problème, il est nécessaire d'accroître la précision des calculs en augmentant l'attribut d'*Oversampling* dans les propriétés du *Solver*.

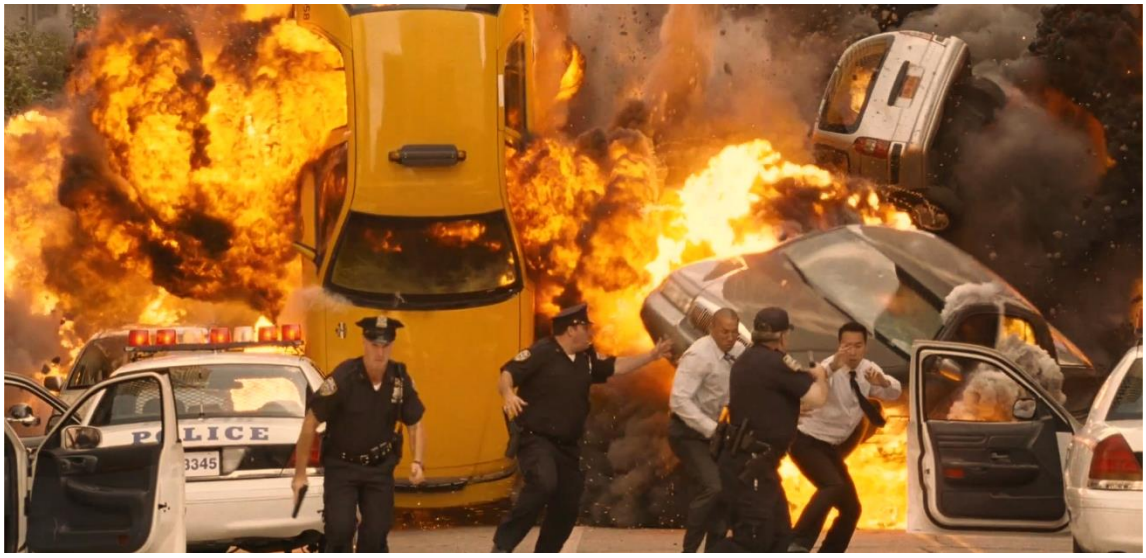
1.5. Les Rigid Bodies

Qu'est-ce que c'est ?

Un *Rigid Body* est un objet dynamique solide possédant des propriétés physiques lui permettant d'interagir avec d'autres objets dynamiques ou d'être affecté pas des champs de force. Utiliser des *Rigid Bodies* peut permettre d'animer des corps soumis à la physique sans avoir à le faire à la main. Ainsi, en appliquant à ces objets une gravité et en paramétrant notamment leur masse, on va pouvoir simuler le comportement d'objets qui tombent ou interagissent avec d'autres éléments.

À quoi ça sert ?

Dans le cas d'effets pyrotechniques, on peut imaginer utiliser cette technique pour animer des objets éjectés ou simplement déplacés par le souffle d'une explosion, tels que des voitures ou autres éléments de décors. Il serait également possible de s'en servir pour réaliser de gros débris lors de ce genre d'effet. L'utilisation de *Rigid Bodies* pour de telles simulations donne des résultats beaucoup plus précis qu'en utilisant des instances de particules. En effet, chaque débris va réagir différemment en fonction de sa forme et de ses propriétés physiques. Aussi, comme nous l'avons déjà évoqué, contrairement à une instance de particule, l'utilisation de *Rigid Bodies* va nous permettre d'éviter les interpénétrations lors de collisions.



Avengers, Joss Whedon, 2012.

Comment ça marche ?

Un *Rigid Body* peut être soit un *NURBS* soit un polygone. La seule chose à vérifier avant sa création est le sens de ses normales car les collisions ne peuvent s'effectuer que du côté où se trouve la normale. Par conséquent, il vaut mieux éviter de convertir une surface plane en *Rigid Body* car elle ne réagirait pas correctement en termes de collisions. Il faut distinguer tout d'abord deux types de *Rigid Bodies*, les passifs et les actifs. Un *Rigid Body* passif est un objet physique fixe pouvant être animé qui va permettre de transmettre des informations de collision. Un *Rigid Body* actif, en revanche, est un objet physique mobile qui ne pourra pas être animé car il dépend simplement de calculs dynamiques. Il sera cependant possible d'interagir avec lui par le biais d'autres *Rigid Bodies* passifs ou actifs, ou à l'aide de champs de force ou de particules. Étant donné qu'il s'agit d'un objet physique, un *Rigid Body* actif possède une série de propriétés telles qu'une masse, un centre de gravité, un indice de friction, etc. Il va ainsi nous être possible de simuler tout type d'objet, de poids et matériau différent. Pour la réalisation de débris rocheux, par exemple, des *Rigid Bodies* pourraient être paramétrés avec une masse importante et beaucoup de friction afin qu'une fois au sol ils s'immobilisent rapidement. Lorsque l'on crée un *Rigid Body*, le premier réflex va être de créer une gravité afin que son comportement ressemble à celui d'un objet sur Terre, à moins que l'on souhaite simuler un corps dans l'Espace. Connecter une gravité à un *Rigid Body* est si courant que si l'on sélectionne une géométrie et que l'on crée une gravité, la géométrie va automatiquement être convertie en *Rigid Body*.

Contrairement à des instances de particules, une simulation de dizaines de *Rigid Body* peut rapidement s'avérer coûteux en temps de calcul. C'est pourquoi nous avons accès à toute une série de paramètres nous permettant d'accélérer la simulation. Premier attribut intéressant, *Collision Layer*. Sachant que la partie la plus conséquente des calculs concerne les collisions, il serait dommage d'effectuer des tests de collision inutiles et de gâcher ainsi de la puissance de calcul. L'attribut *Collision Layer* permet de

choisir pour chaque *Rigid Body* dans quel calque de collision il se trouve, et ainsi effectuer des tests de collision uniquement avec les autres objets dynamiques se trouvant dans ce même calque. Cela peut être utile si deux explosions ont lieu côte à côte et que chacune d'elle génère un groupe de débris qui n'interagissent pas entre eux. Si chaque système de débris est placé dans un calque de collision différent, le temps de calcul peut être divisé par deux. Il existe d'autres paramètres permettant d'accélérer le calcul des collisions. *Stand In*, par exemple, permet de remplacer la forme de l'objet qui entre en collision par une primitive — l'aspect visuel du *Rigid Body* n'en est pas affecté pour autant. *Step Size* permet de régler la fréquence à laquelle Maya effectue ses tests de collision. Si les débris ne se déplacent pas très rapidement, il est possible d'augmenter légèrement ce paramètre sans poser trop de problème et ainsi baisser le temps de calcul. Il est également possible de changer la méthode de calcul affectée aux *Rigid Bodies* à l'aide du paramètre *Solver Method*, afin d'en choisir un moins coûteux en temps de calcul comme le *Runge Kutta* ou le *Mid Point*. Les collisions seront alors beaucoup moins précises mais la vitesse de lecture sera beaucoup plus rapide. Cela pourrait, par exemple, être utile pour mettre en place un effet rapidement. Une fois que la simulation nous conviendrait nous pourrions alors retourner à la méthode de calcul la plus précise pour affiner les paramètres et lancer le cache et le rendu.

1.6. Les Fluides

Qu'est-ce que c'est ?

Un *Fluide* ***Maya*** est un système dynamique implémentant la mécanique des fluides numériques qui permet de simuler et de rendre une grande variété d'effets volumétriques liquide ou gazeux. Pour pouvoir exister et s'étendre, un Fluide nécessite un environnement 3D. Étant donné qu'il serait impossible d'utiliser un espace infini pour des raisons de temps de calcul, un Fluide va évoluer à l'intérieur d'un espace cubique fermé appelé conteneur. Il est possible d'utiliser les bords de ce conteneur comme des murs virtuels pour notre Fluide ou simplement pour limiter l'affichage et le calcul de notre simulation. Un Fluide est composé de *Voxels*. Sa qualité est liée à sa résolution, c'est-à-dire son nombre de *Voxels*.

À quoi ça sert ?

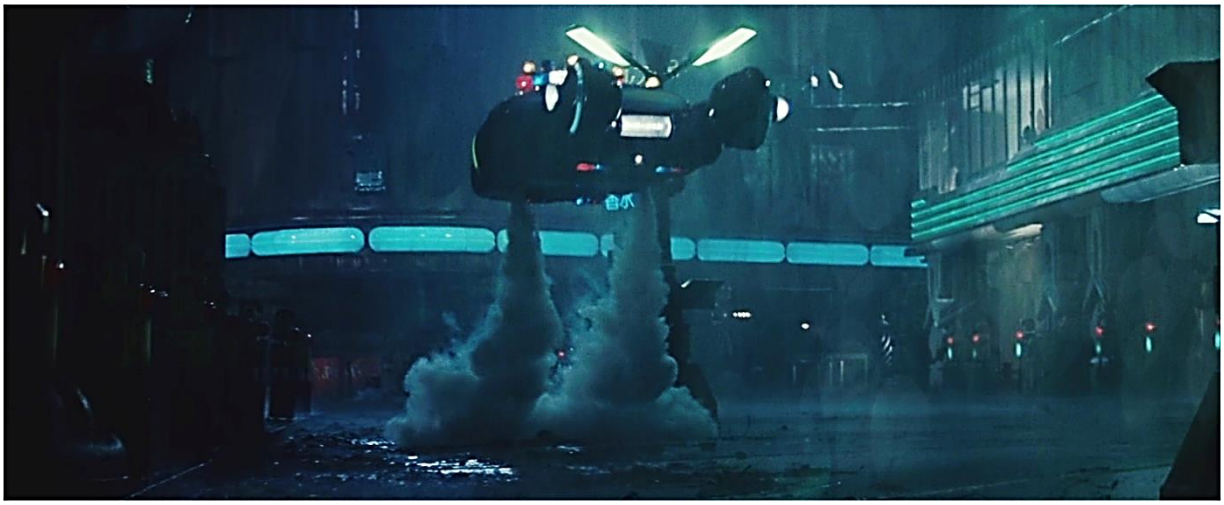
L'utilisation de Fluides va nous permettre de réaliser de nombreux effets pyrotechniques principaux mais aussi toute sorte d'effets secondaires. Premier effet conséquent, le feu. En effet, le système de simulation de Fluide est parfait pour réaliser ce genre d'effet. Il est considéré, néanmoins, comme étant l'un des plus compliqué car le paramétrage est très précis, notamment au niveau du rendu, et il n'existe pas vraiment de méthode miracle pour faire un beau feu. De plus, il existe plein de types de feux différents et il faudra revoir ses paramètres pour chaque cas de figure, en fonction de la taille, de l'éclairage, du type de combustible, etc.

Le Hobbit : Un Voyage Inattendu, Peter Jackson, 2012.

Battleship, Peter Berg, 2012.

Autre application évidente dans la réalisation d'effets pyrotechniques, les explosions. En effet, grâce à des propriétés semblables à ce que nous connaissons dans la pyrotechnie réelle, les Fluides ***Maya*** vont nous permettre de simuler des réactions chimiques basées sur un combustible (gaz, essence, bois…), un comburant (dioxygène de l'air) et une énergie d'activation (flamme, étincelle, chaleur…). C'est donc à partir de ce triangle du feu que nous allons pouvoir simuler des réactions explosives et la combustion de différents matériaux. Un Fluide ***Maya*** peut ainsi se composer d'air (*Density*), de chaleur (*Heat*) et de carburant (*Fuel*), éléments propice à cette réaction.

Enfin, nous allons également pouvoir nous servir de ce système dynamique pour la génération d'effets secondaires tel que de la fumée, de la poussière, du sable, etc. La différence est que nous n'utiliserons, à priori, pas de système d'ignition car nous avons besoin simplement d'une masse unicolore se déplaçant dans l'air avec une certaine perturbation. Il est possible, néanmoins, d'utiliser du carburant et de la chaleur pour réaliser ce genre d'effets mais simplement dans le but de contrôler la façon dont le fluide réagit et non pas au niveau du rendu pour l'incandescence par exemple.



Blade Runner, Ridley Scott, 1982.

Comment ça marche ?

Lorsque l'on réalise des effets à l'aide de Fluides *Maya*, il faut distinguer deux aspects principaux, la simulation et le rendu. Bien que ces deux étapes soient tout aussi importantes l'une que l'autre, il faut veiller à les faire dans le bon ordre. En effet, la simulation est la première chose à régler et ce n'est qu'une fois que la dynamique du *Fluide* est convaincante que l'on peut passer au *Shader*. Ainsi, si une simulation n'est pas satisfaisante, un bon *Shader* ne pourra pas la rattraper. À l'inverse, modifier la simulation pourra changer considérablement l'aspect visuel d'un Fluide. Ce serait donc une perte de temps de venir régler avec précision les paramètres de rendu avant que la simulation soit bonne. Ceci étant dit, intéressons-nous donc à la première étape, la simulation.

Afin d'émettre un Fluide nous avons besoin de deux choses, un conteneur et un émetteur. Chacun de ces deux objets possèdent des paramètres propres. L'émetteur va posséder des attributs liés à la génération du Fluide comme la vitesse initiale, la quantité de Fluide émise, la turbulence d'émission, etc. Le conteneur pourra, quant à lui, être paramétré au niveau des caractéristiques générales du Fluide, c'est-à-dire concernant l'évolution du Fluide après son émission. Comme nous l'avons évoqué précédemment, un Fluide peut être constitué de différents composants qui vont être liés entre eux. Le premier composant est la densité (*Density*). Il s'agit de la matière du Fluide. Il pourrait être comparé au dioxygène contenu dans l'air. Le deuxième composant est la température (*Temperature*). Il est possible d'en créer en générant de la chaleur (*Heat*) depuis un émetteur ou à la suite d'une réaction explosive. Ce composant correspond à la partie incandescente du Fluide. Enfin le dernier, le carburant (*Fuel*), correspond au combustible. Il permet à la chaleur de se propager en même temps qu'il est consumé. Tous ces composants vont pouvoir être émis en même temps ou séparément en fonction de l'effet recherché. Il est également possible de se passer des certain d'entre eux dans certains cas de figure. Pour simuler une fumée, par exemple, il est possible de

se servir uniquement de la densité et d'utiliser les paramètres de rendu pour lui donner différents aspects visuels.

Afin de contrôler le comportement des différentes composantes d'un Fluide, il existe, dans les paramètres du conteneur, une sous-section réservée à chacun de ces composants qui regroupe une série d'attributs. Il est donc possible de les paramétrer individuellement. Il existe des attributs similaires sur différents composants, notamment entre la densité et la température. Les attributs individuels, selon moi, les plus importants pour commencer à régler son Fluide, et que nous allons constamment venir ajuster, sont la *Buoyancy*, la *Dissipation* et la *Diffusion*. La *Buoyancy* correspond à la légèreté du composant, c'est-à-dire à la vitesse à laquelle il s'élève dans le conteneur. La *Dissipation* est à la vitesse à laquelle le composant va disparaître — se dissiper. Enfin le paramètre de *Diffusion* permet de régler la capacité du composant à s'étendre dans l'espace au fil du temps. Si nous souhaitons simuler un feu, par exemple, il va falloir doter notre Fluide d'une forte chaleur avec peu de dissipation dans la température mais une forte *Buoyancy* car il faut que la simulation s'élève rapidement. De plus, la densité — qui correspond au composant qui va, par défaut, gérer l'opacité du fluide — devra avoir, quant à lui, une dissipation assez forte afin que les flammes soient assez courtes. Si, en revanche, nous souhaitons avoir, en plus des flammes, de la fumée, il va falloir faire l'inverse et donner une dissipation assez forte au niveau de la température et assez faible au niveau de la densité. De cette façon, la chaleur — la partie jaune/rouge de la simulation — se consume très vite, ne s'élève pas trop haut et fait place à de la fumée — blanc par défaut — qui s'élève dans les airs.

Autre attribut majeur dans une simulation de Fluide, le *Swirl*. Il correspond à la turbulence générale du Fluide. En effet, contrairement à ceux dont je viens de parler, cet attribut va permettre de contrôler tous les composants en même temps car il altère une composante dont je n'ai pas encore parlé, la vitesse (*Velocity*). En effet, en plus de la *Density*, la Temperature et le *Fuel*, la *Velocity* est un composant du Fluide qui est

constitué de vecteurs servant à déplacer tous les autres composants. Chaque *Voxel* va ainsi être doté d'une vitesse et d'une direction qui va pousser le Fluide vers les *Voxels* des alentours. Lorsque l'on augmente le *Swirl*, on ajoute de la turbulence dans la vitesse et donc dans tous les composants du Fluide. Le résultat est une simulation avec plus de détails, d'imperfections, de mouvements, donc de réalisme, de crédibilité. C'est un paramètre très important pour une simulation de feu, par exemple, car c'est lui qui va donner cette sensation de chaos dans le mouvement des flammes. Notons qu'il est également possible d'ajouter de la turbulence uniquement sur la température si cela est nécessaire grâce à l'attribut *Turbulence* accessible dans l'onglet *Temperature*.

Une fois que notre simulation est satisfaisante, nous pouvons passer à la deuxième étape, tout aussi importante, le *Shading* de notre *Fluide*. Cette étape correspond aux réglages visuels du *Fluide* — l'opacité, la couleur, l'incandescence, la texture, etc. Tout d'abord, l'opacité. C'est à partir de ce paramètre, que nous allons pouvoir sculpter notre Fluide. Par défaut, l'opacité est connectée à la densité, c'est-à-dire qu'aux endroits où il y a de la densité le fluide est affiché, et là où il n'y en a pas, le fluide est invisible. Il est ainsi possible de venir augmenter l'opacité dans les zones où il y a une certaine valeur de densité et de la réduire dans les zones où il y en a moins afin de donner plus de contraste à notre Fluide, des contours plus nets et un corps plus opaque. Ce genre de réglage peut être intéressant lors de la réalisation d'une explosion nucléaire, par exemple, qui est censé émettre un nuage de fumée aux contours bien découpés. Il est également possible de venir connecter cette opacité à d'autres composants de notre Fluide. Nous pouvons par exemple la connecter à la température, notre simulation se verra alors sculptée en fonction de l'émission de chaleur, ce qui pourrait éventuellement être utile pour la réalisation de flammes sans fumée.

Autre paramètre important, la couleur. Par défaut, la couleur de notre fluide est connectée à un paramètre constant blanc, ce qui signifie que toutes les parties opaques et éclairées du fluide seront blanches. Ce paramètre est intimement lié à la dernière

composante visuelle, l'incandescence. L'incandescence correspond à la partie lumineuse du fluide. Par défaut, il est lié à la température, ce qui signifie que la température va venir diriger la luminescence de notre simulation. Si la couleur de notre fluide est le blanc, il va venir obstruer l'incandescence à moins qu'il ne reçoive pas de lumière. Lors d'un effet pyrotechnique, il est d'usage de choisir une couleur noire ou gris foncées afin de révéler cette incandescence et d'obtenir une fumée sombre.

Enfin, il est possible de connecter ces trois composantes visuelles à une texture que l'on va pouvoir animer afin de donner plus de détails à la simulation. Lors de la réalisation d'un effet de combustion, ajouter une texture sur l'incandescence peut s'avérer ainsi très intéressant voire indispensable afin d'augmenter les détails dans le feu. L'inconvénient de cette technique est que cette texture n'est pas directement liée à la simulation, par conséquent il va falloir l'animer selon l'un des trois axes X, Y ou Z, en respectant la direction que prend notre fluide.

1.7. Le plug-in DMM

Qu’est-ce que c’est ?

Toujours afin d'aller plus loin dans la réalisation d'effets pyrotechniques, je me suis penché sur DMM — Matière Moléculaire Digitale —, un plug-in *Maya* permettant de réaliser des destructions et des déformations de géométrie. Contrairement à d’autres systèmes de simulation traditionnels, ce plug-in utilise une méthode appelée FEM — Méthode des éléments finis — qui est une technique d’analyse numérique permettant de résoudre des calculs complexes dans le but de représenter le comportement dynamique de systèmes physiques variés. Son utilisation est assez simple mais peut s’avérer capricieuse dans certaines situations, notamment lorsque l’on commence à travailler sur des géométries complexes ou avec un grand nombre de polygones.

À quoi ça sert ?

En ce qui concerne ses applications pour de la pyrotechnie de synthèse, cet outil nous sera évidemment utile pour altérer des surfaces de différentes matières lors de l'interaction avec un effet assez puissant. On pourrait imaginer s'en servir lors d'une explosion pour détruire un mur de brique ou de béton, des fenêtres, un toit en tôle, une porte en bois, etc. Il serait à priori également possible de simuler, par exemple, un objet en plastique ou une feuille de papier qui se déformerait sous l'effet de la chaleur d'un feu. Enfin, nous pourrions aussi imaginer déformer des surfaces suite à des retombées d'objets ou de débris projetés après une explosion. Bien que de nombreuses autres applications m'échappent surement, les usages, bien que variés, sont fondamentalement voués à un même but, déformer et détruire des objets.

Avengers*, Joss Whedon, 2012.*

Comment ça marche ?

Lorsque l'on souhaite déformer un objet à l'aide de ce plug-in, la première étape est de le convertir en objet DMM. Mais avant de pouvoir le faire, il faut s'assurer que l'objet soit bien fermé. Autrement dit, pour qu'un objet puisse être converti, il faut que toutes ses arrêtes soient utilisées par deux polygones — ou deux faces. Il n'est, par exemple, pas possible de convertir un plan en objet DMM. Cette contrainte est en fait basée sur un principe physique somme toute assez logique selon lequel tout objet physique, même le plus fin, possède une épaisseur. De plus, lors de la destruction de la géométrie, DMM va recréer des faces internes afin de donner l'illusion que l'objet est consistant, qu'il est fait de matière.

Pour caractériser cette matière, nous avons accès à une série de paramètres représentant toutes les caractéristiques physiques que peut avoir un objet destructible ou déformable — masse, résistance, flexibilité, etc. Afin de ne pas repartir de zéro à chaque fois, DMM nous nonne accès à une série de configurations prédéfinies (*Preset*) allant du métal au bois en passant par le cristal. Néanmoins, il est rare d'utiliser un *Preset* sans lui apporter de modifications. L'attribut *Youngs*, par exemple, gère la rigidité du matériau, l'attribut *Young Damp* sa capacité à se stabiliser, et l'attribut *Toughness* sa résistance. Imaginons que l'on souhaite détruire le pare-brise d'une voiture qui reçoit un débris. Le paramètre *Youngs* devra être suffisamment élevé pour que le pare-brise ne s'affaisse pas comme un tissus au moment de l'impact, mais pas trop car il doit garder une certaine souplesse. *Young Damp* devra être suffisamment élevé pour que le pare-brise s'immobilise rapidement après la collision ; néanmoins, si cette valeur trop haute, le pare-brise ne se cassera pas suffisamment. Enfin, *Toughness* devra être assez élevé pour que l'objet ne se brise pas en mille morceaux lors de l'impact.

Étant donné qu'une géométrie DMM est un objet entièrement dynamique, lorsque l'on lance la simulation, il va tomber avec la gravité, comme un simple *Rigid*

Body. Il existe donc des outils permettant de le maintenir à une position précise. Le premier s'appelle *Passive Region*, il permet de créer un cube d'attache pour notre géométrie. Toute partie de l'objet DMM se retrouvant à l'intérieur de se cube deviendra inactive. Un autre outil très intéressant, *Glue Region*, permet de crée un cube qui agit comme de la colle entre deux objets DMM. À la différence de *Passive Region*, avec cet outil, l'objet accroché peut se décoller s'il est soumis à une force suffisante. Cela peut être utile, par exemple, dans le cas de notre pare-brise, pour le faire se détacher littéralement de la carrosserie, si le débris reçu possède un poids conséquent. Enfin, il existe un outil, toujours matérialisé sous forme d'un cube, qui permet de subdiviser toute partie de l'objet DMM se trouvant à l'intérieur — la *Density Region*. Cela peut être assez utile si l'on souhaite fracturer seulement une partie d'un objet. Le reste de l'objet restera alors en plus basse définition, ce qui accélèrera la vitesse de calculs.

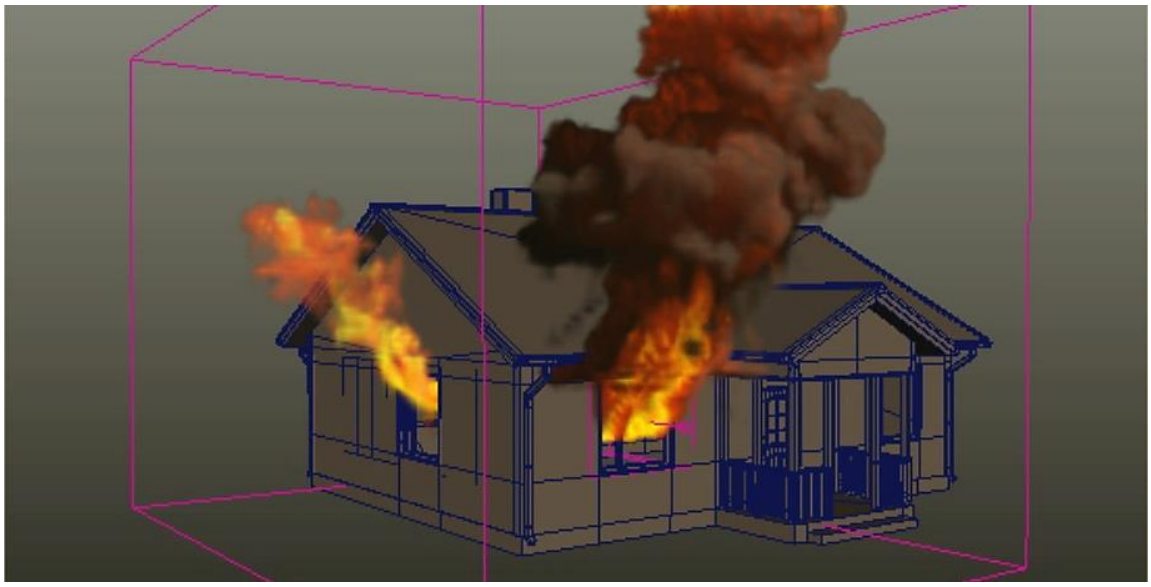
PARTIE 2

Expérimentations et Projets Simples

2.1. Expérimentations

Dans cette sous-partie, je présente mes recherches qui m'ont semblés les plus pertinentes, à la fois car elles m'ont apportés personnellement certaines choses, mais aussi parce qu'elles servent, d'une manière ou d'une autre, ma problématique.

2.1.1. Incendie domestique



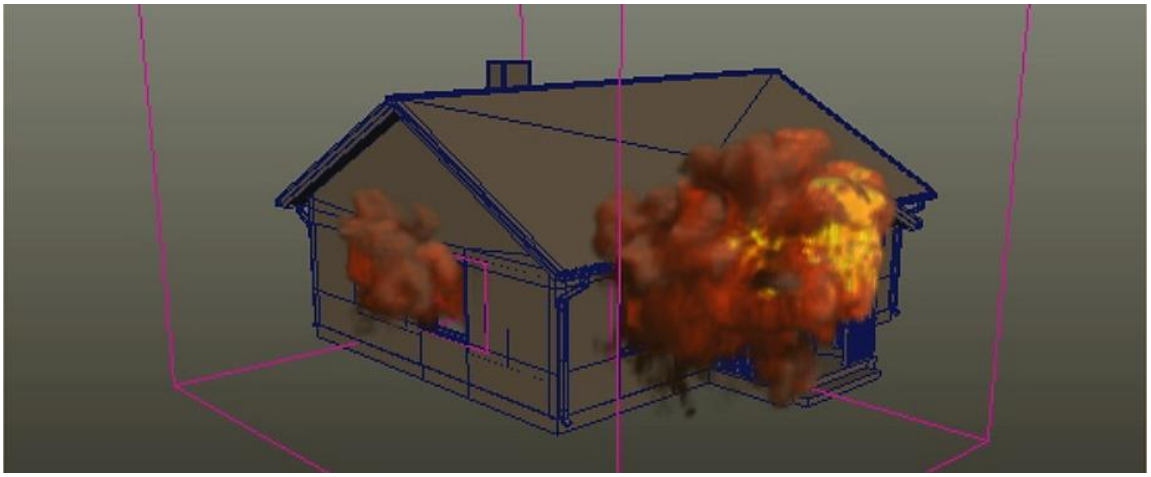
Dans cette première série des tests, j'ai tenté de comprendre comment mettre le feu à une maison. J'ai récupéré un modèle de maison 3D gratuit sur internet afin de pouvoir m'en servir comme base pour l'échelle et les collisions.

Après avoir installé mon conteneur autour de la maison et placé son émetteur à l'intérieur, je m'aperçois rapidement que je ne vais pas pouvoir utiliser cette géométrie comme objet de collision. Premièrement la maison est faite en un seul bloc, donc lorsque le fluide est généré à l'intérieur il ne peut pas remplir tout l'espace tant il est vaste et se retrouve bloqué au plafond. Deuxième chose, les murs étant de simples plans sans épaisseur, lorsque le fluide entre en collision avec, il le traverse partiellement. Pour régler ce problème j'ai donc recréé une géométrie permettant de confiner le fluide dans un espace déduit, loin des murs.

Aussi, émettre simplement le Fluide à l'intérieur de la pièce n'avait pas suffisamment d'impact. J'ai ainsi préconisé l'utilisation de vitesse initiale en plaçant mes émetteurs directement aux fenêtres pour que la fumée soit directement émise là où j'en avais besoin. Pour la simulation, j'ai joué avec les paramètres de chaleur et de

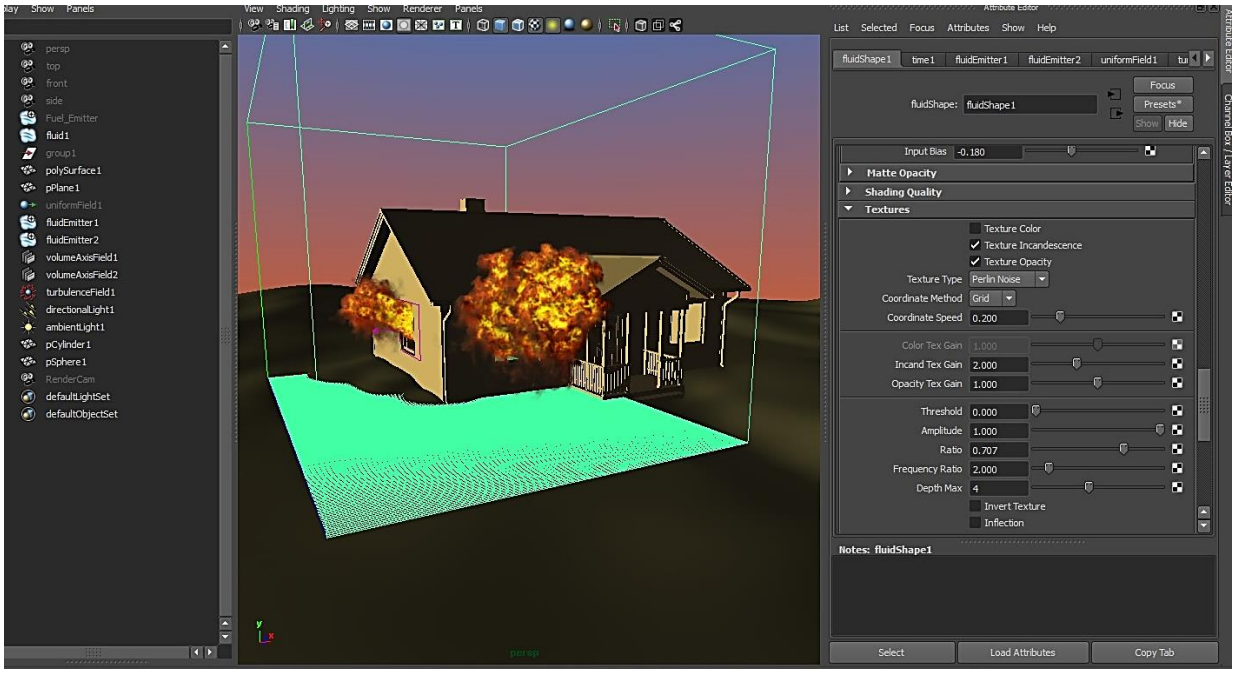
carburant pour essayer de donner un peu de variation et de créer quelques petites volutes pour donner un peu plus de vie à l'effet.

Une fois que j'ai pu obtenir une simulation plutôt convaincante, j'ai voulu aller plus loin, en tentant de créer une explosion avant l'émission de fumée. La tâche s'est alors compliquée car je n'arrivais pas à donner suffisamment de force au fluide pour le pousser hors de la maison avec assez de vitesse et d'entrain. Je suis finalement arrivé à un compromis intéressant en utilisant des champs de force que j'ai animés pour pousser le fluide au moment de la détonation.



Après plusieurs simulations relativement convaincantes, je suis tombé des nues lorsque j'ai voulu passer au rendu. En effet, quand il s'agit de faire des particules, le résultat est directement visible, la plupart du temps en temps réel. Il y a donc beaucoup moins de surprises. Les fluides, quant à eux, sont plus capricieux. En effet, une fois que la simulation nous convient, il faut augmenter la résolution du conteneur pour que le rendu soit meilleur. Le problème est qu'il va donc falloir changer tout un tas de paramètres pour que le fluide réagisse de la même façon. Et même après avoir passé cette étape, reste le délicat exercice de *Shading* et de texture qui peut vraiment changer la donne dans un sens ou dans l'autre. Une simulation qui manque un peu de détail peut se voir améliorée grâce à des textures et un bon *Shading*. Cependant, s'il est mal paramétré, un fluide avec une bonne simulation peut s'avérer très moche au rendu. Le paramètre d'opacité du fluide, par exemple, est un élément assez délicat. En effet, il

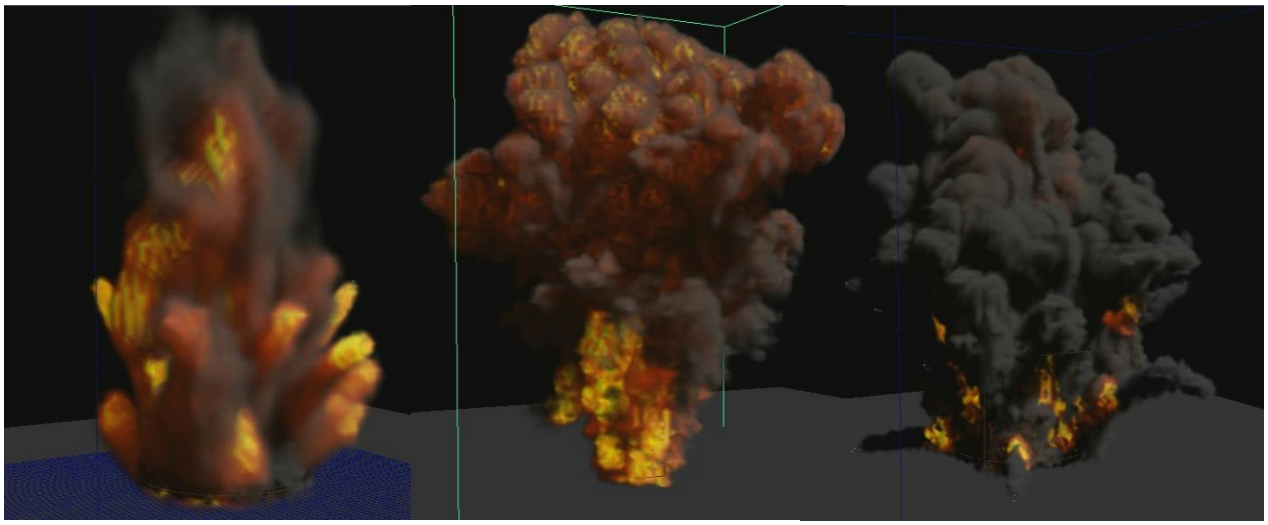
peut permettre de venir redonner du détail aux contours d'une simulation. Le problème est qu'il peut en même temps venir révéler des défauts de contours.



Pour l'explosion de ma maison, après avoir mis en cache une simulation que je trouvais intéressante, j'ai passé beaucoup de temps à tenter de régler les textures, le *Shading* et précisément l'opacité du fluide sans me rendre compte que je cherchais dans la mauvaise direction. En effet, dans ma simulation, l'explosion émise depuis l'intérieur génère de la densité, de la chaleur et du carburant. Ainsi, lorsque l'explosion apparait, elle contient non seulement des zones d'incandescence visibles mais aussi des parties de fumée noire sur ses contours. Le problème est qu'au moment précis de l'explosion, je souhaiterais n'avoir encore aucune fumée noire, seulement un fluide allant du blanc au rouge et que la fumée n'apparaisse qu'une fois l'explosion passée. Ma simulation semblait me convenir. J'ai donc passé du temps à chercher le paramètre qui me permettrait de réaliser cet effet directement dans le *Shading*. J'ai donc tenté de réduire l'opacité des contours afin d'éliminer les parties noires. Cela n'a fait que rajouter des problèmes au niveau des contours. J'ai ensuite tenté de régler la couleur de l'incandescence pour que le fluide soit rouge également dans mes parties où il n'y a pas de température. Le problème était que mon incandescence, plutôt correcte au début,

devenait trop importante à la fin. L'animer n'est pas non plus une solution judicieuse car ce paramètre agit sur tout le fluide, la transition serait étrange. J'ai enfin tenté de jouer avec la texture de mon fluide, d'augmenter la puissance de l'incandescence et de baisser celle de l'opacité, mais aucune modification n'a été bénéfique. Ainsi, après un certain nombre d'essais ratés, j'ai dû me rendre à l'évidence ; je devais refaire ma simulation. Mon émission initiale contient trop de densité et pas assez de chaleur. Le fluide possède alors des zones froides sur ses contours, ce qui crée cette fumée noire. J'ai donc dû augmenter l'émission de carburant au début et la laisser un peu se répandre avant d'émettre de de la chaleur puis de la densité pour que chaque *Voxel* de densité puisse immédiatement rencontrer de la chaleur et donc que sa couleur dépende uniquement de l'incandescence au moment de l'explosion.

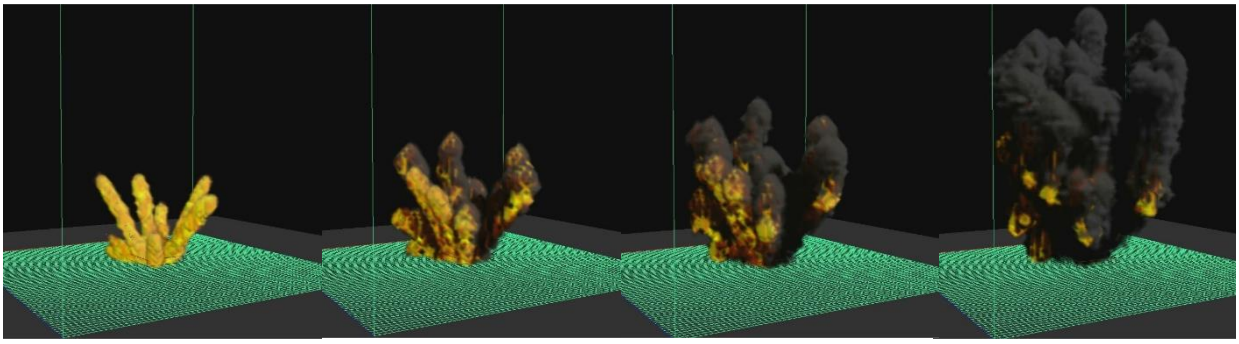
2.1.2. Explosions de boules de feu



Dans cette autre série d'expérimentations, je me suis intéressé à l'émission de fluide depuis des particules dans le but de réaliser des explosions.

Il s'est avéré que cette technique possède de véritables avantages à condition de l'utiliser correctement. Au vue des différents tests que j'ai pu effectuer, je pense avoir trouvé une technique qui donne des résultats assez intéressants. Elle consiste à émettre un petit nombre de particules avec une forte vitesse initiale en leur affectant une gravité pour qu'elles retombent ensuite sur le sol. Chacune de ces particules est ensuite utilisée comme émetteur pour un autre groupe de particules qui sera utilisé comme source d'émission pour le Fluide. Il est important de paramétrer cette seconde émission de particules, de telle sorte qu'elles aient une faible durée de vie, afin de ne pas se retrouver avec des trainées de fumée persistantes. L'émetteur de Fluide connecté aux particules est ensuite paramétré pour émettre les composants nécessaires. J'utilise en général les trois — densité, température et carburant — afin de générer des boules de feu laissant une trainée de fumée derrière elles.

Durant mes différents essais, j'ai pu constater que, lorsque la simulation de fluide réagit de façon étrange, la simulation de particules est presque toujours en cause. Par conséquent, il est important de prendre du temps pour la régler correctement afin de ne pas avoir de surprises au moment des calculs de fluide. Il m'est arrivé lors d'un test de créer un système de particules qui perdait rapidement sa vitesse avant de retomber sur le sol. Ainsi, les particules n'ayant pratiquement plus de vitesse à un certain moment, le fluide n'avait plus rien à suivre et se mettait à s'élever brutalement dans les airs. Un résultat visuellement très étrange.



Un des avantages de ce système, est qu'il est très rapide de mettre en place une simulation de particules. Ainsi, ce peut être un gain de temps non négligeable lorsqu'il s'agit de paramétrer l'émission du fluide. De plus, la facilité avec laquelle nous pouvons contrôler des particules peut nous permettre de créer des simulations beaucoup plus complexes qu'avec un émetteur de fluide normal.

L'inconvénient, en revanche, est qu'il faudra généralement augmenter le nombre de calculs effectués par seconde pour notre simulation de fluide, notamment si les particules vont très vite, autrement le fluide pourrait tout simplement ne pas être calculé sur certaines images. Personnellement, je ne pense pas qu'il s'agisse d'un réel problème, car, quoi qu'il arrive, la précision des calculs devra être augmentée tôt ou tard, avant de passer au rendu final.

2.1.3. Le feu

Rendu de flammes réalisées à l'aide d'un fluide 2D.

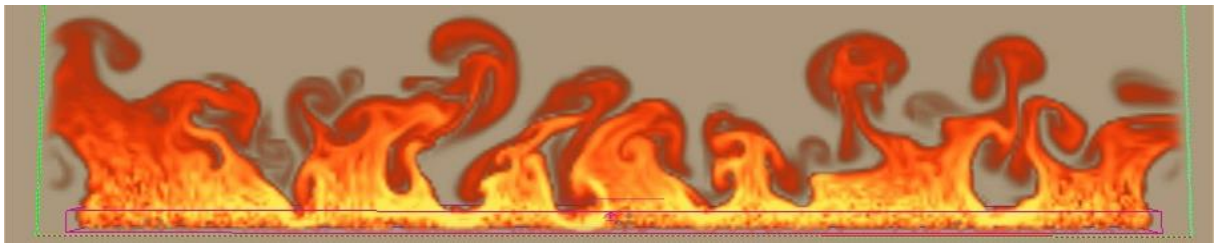
Cette nouvelle série de tests a été consacrée à la réalisation d'une étendue de feu. Il existe autant de type de feu qu'il existe de combustible, de condition climatique, d'environnement, etc. La taille est aussi un paramètre constamment à prendre en considération car une flamme de feu de camp d'un mètre, par exemple, n'aura pas grand-chose à voir avec un feu de forêt de dix mètres. Dans mon cas, il ne s'agit pas d'une petite flamme mais plutôt d'un feu qui apparaitrait, par exemple, suite à la combustion d'une flaque d'essence.

Afin d'avoir le dispositif le plus réactif possible afin de tenter d'apprivoiser cette technique, j'ai décidé d'essayer de créer du feu à partir d'un conteneur de fluide en deux dimensions, beaucoup plus rapide en terme de calculs.

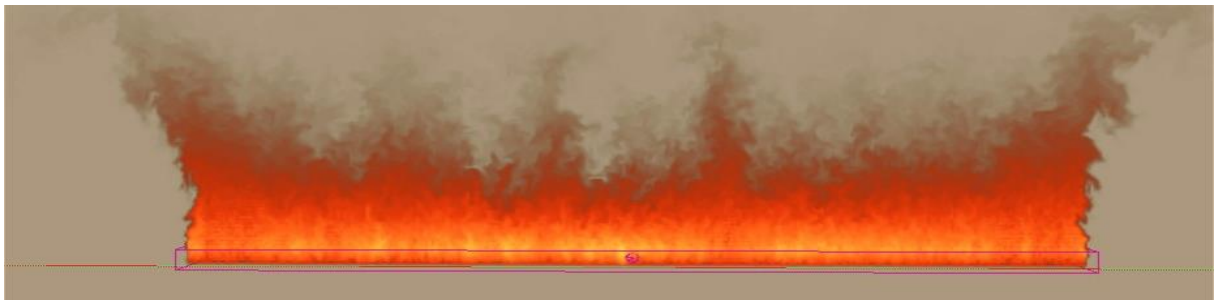
Après avoir effectué de nombreuses expérimentations, je me rends compte que deux paramètres liés à la vitesse sont cruciaux dans la dynamique de ce genre d'effet ; le *Swirl* et le *Noise*. Concrètement, le *Swirl* crée des volutes comparable à de la turbulence, mais d'une manière plus propre à un fluide, tandis que le *Noise* crée une perturbation de type fractale, beaucoup plus aiguisée, qui va venir perturber les contours du fluide. C'est en dosant la valeur de ces deux attributs que nous allons pouvoir transformer une simulation en un fluide qui pourrait s'apparenter à du feu. Trop peu de *Swirl* ou de *Noise* donnera une simulation trop uniforme. En revanche une valeur trop élevée de l'un des deux paramètres créera un mouvement trop caricatural. J'ai ajouté ci-dessous quelques images permettant d'illustrer les différents réglages.

Il va sans dire qu'il existe de nombreuses façons dissimuler un feu. Je peux, par conséquent, dire que je n'ai fait, pour le moment, qu'effleurer la surface de toutes les subtilités de cette technique. D'avantage de découvertes à ce sujet sont à suivre, je l'espère.

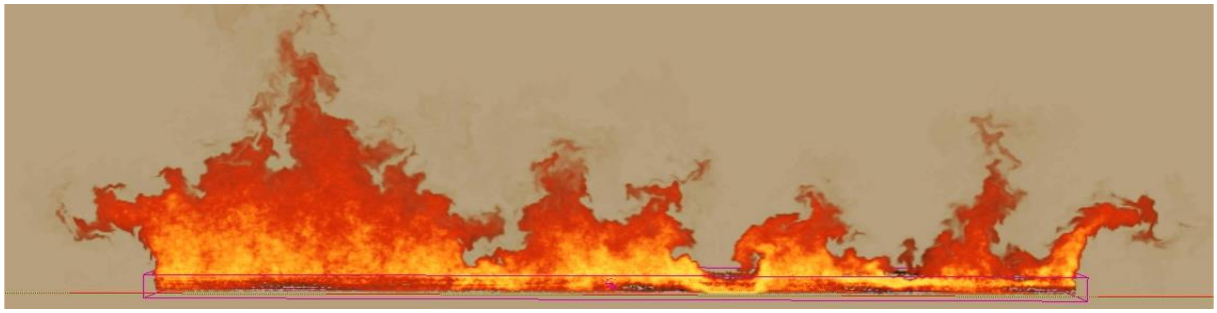
Trop de **Noise**, pas assez de **Swirl**.



Trop de **Swirl**, pas assez de **Noise**.



Trop peu de **Noise** et de **Swirl**.

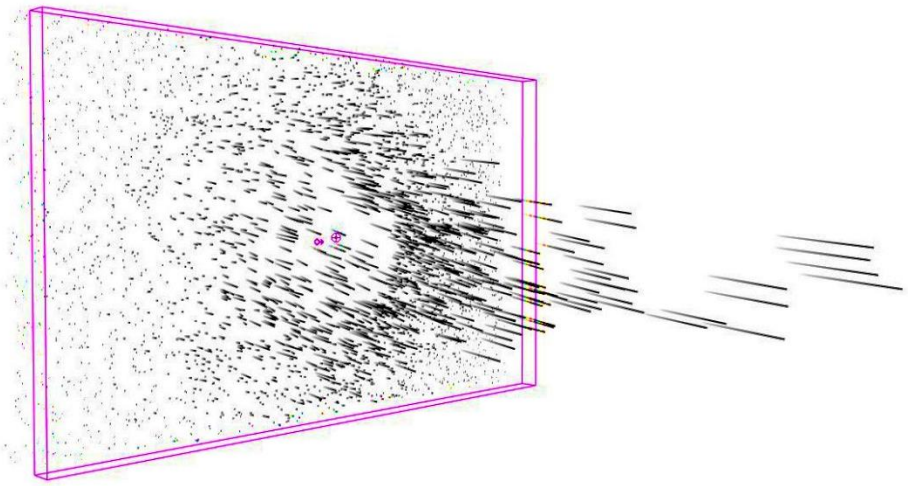


Mélange équilibré entre **Noise** et **Swirl**.

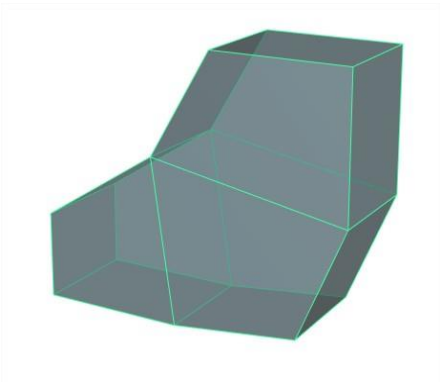
2.1.4. La vitre brisée

Les prochaines pages sont consacrées à la présentation pas à pas d'une étude rapide réalisée autour des techniques de destruction d'une vitre.

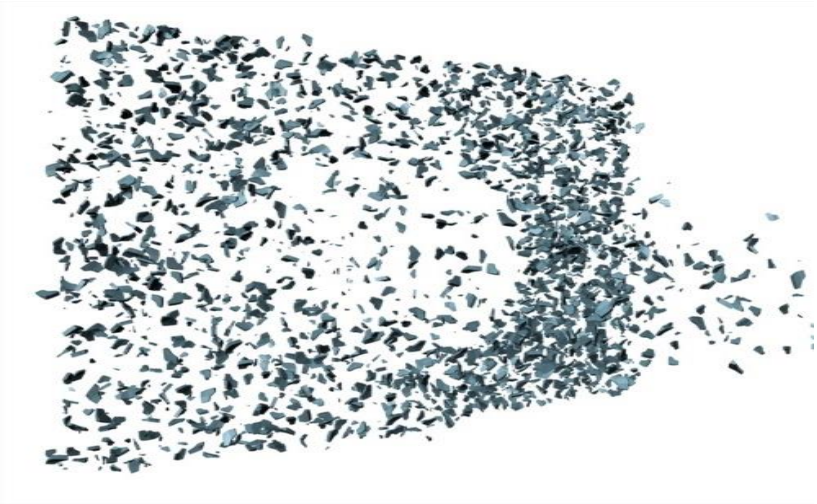
La première technique sur laquelle je me suis penché consiste en la création d'un système de particules qui va être remplacé par des instances afin de simuler une surface de débris de verre qui vole en éclat. Je me suis alors focalisé sur la création de particules à partir d'un volume en forme de rectangle pour simuler la surface d'une fenêtre. J'ai ensuite créé une sphère animée que j'ai fait entrer en collision avec les particules pour créer un effet d'éclatement par le centre.



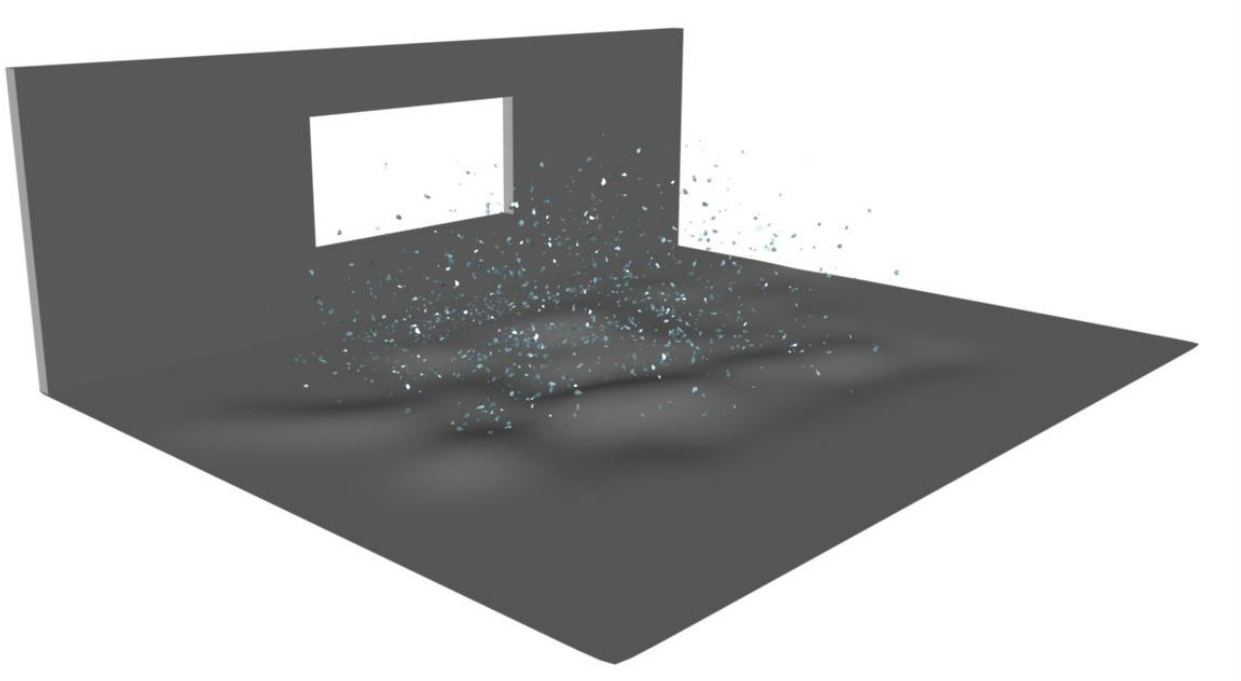
J'ai ensuite remplacé chaque particule par une seule et même géométrie à l'aide du système d'instances. À chaque instance a ensuite été affecté une taille, inclinaison et vitesse de rotation différente afin de créer une simulation moins uniforme.



Le résultat obtenu est intéressant mais ne conviendra pas dans un plan rapproché pour plusieurs raisons. D'abord, il ne permet de simuler la destruction progressive d'une vitre car le système est déjà composé d'une série de fragments. Ensuite, étant donné que chaque morceau est en fait calculé comme étant un simple point dans l'espace, les collisions des fragments avec une autre surface seraient le théâtre d'imprécisions de rebonds et d'interpénétrations des surfaces.



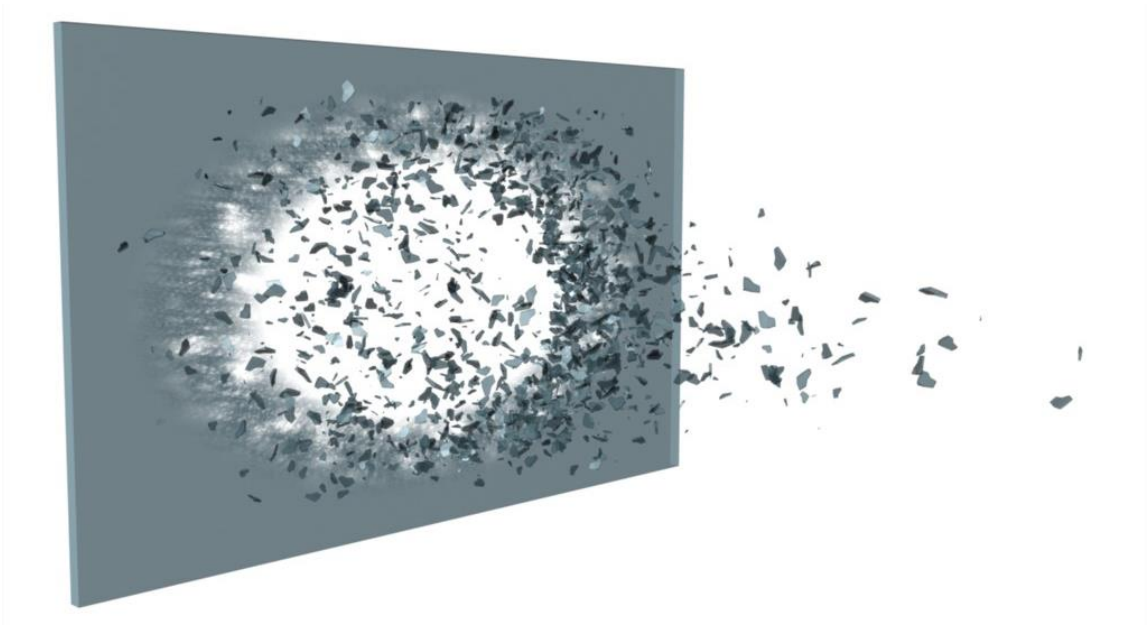
Néanmoins, lorsque j'effectue des tests rapide de rendu, je me rend compt qu'en plan large, l'effet fonctionne plutôt bien.



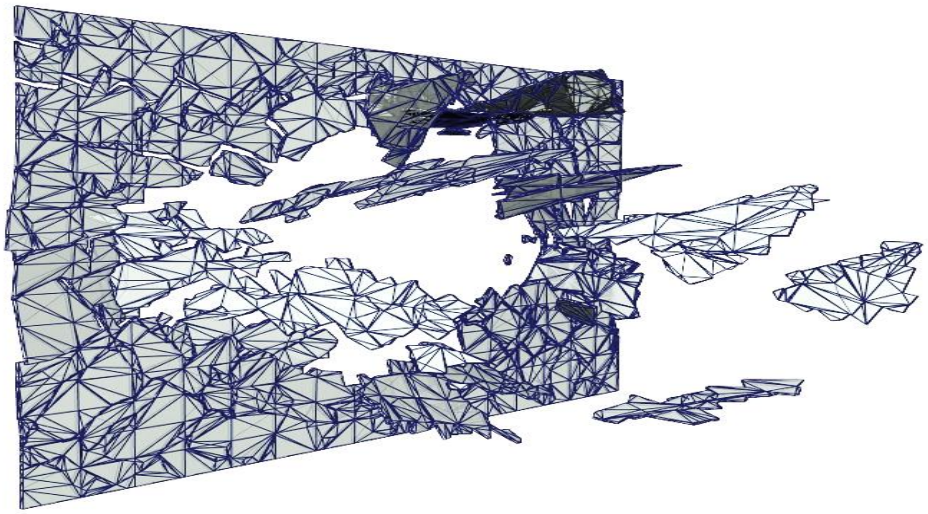
Si l'on souhaite, malgré tout, simuler l'explosion progressive de la vitre, il est possible de le faire en n'affichant uniquement les débris qui sont en mouvement. A l'aide d'une simple expression dans l'opacité des particules, j'ai pu réussir à créer une simulation de débris de verre qui apparaissent à mesure que les particules sont éjectées.

J'ai ensuite créé un rectangle avec une texture animé dans la transparence afin

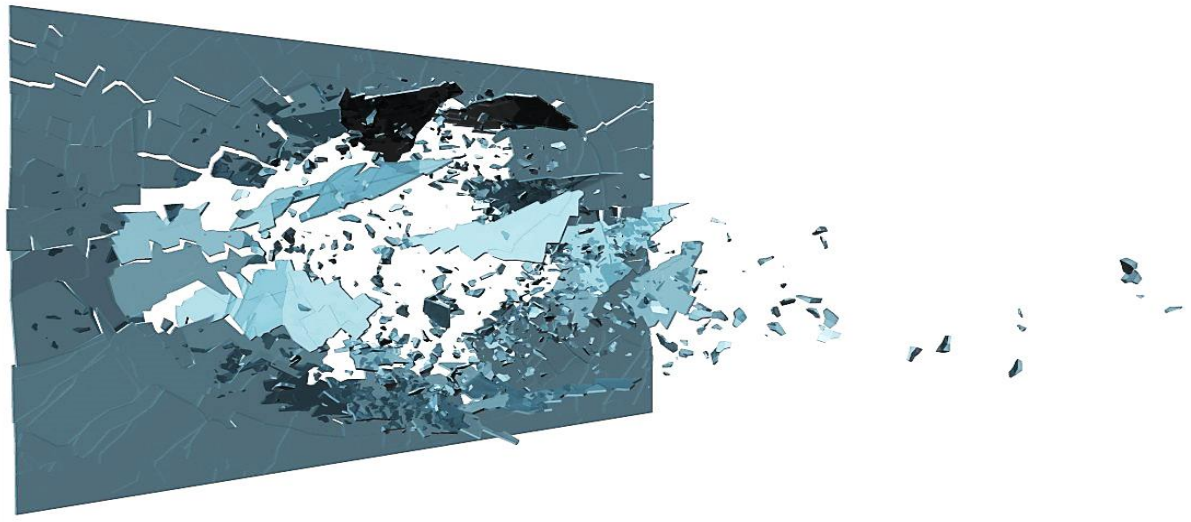
de tenter de simuler rapidement l'effet de désintégration progressive de la vitre. Inutile de dire que le résultat laissait quelque peu à désirer.



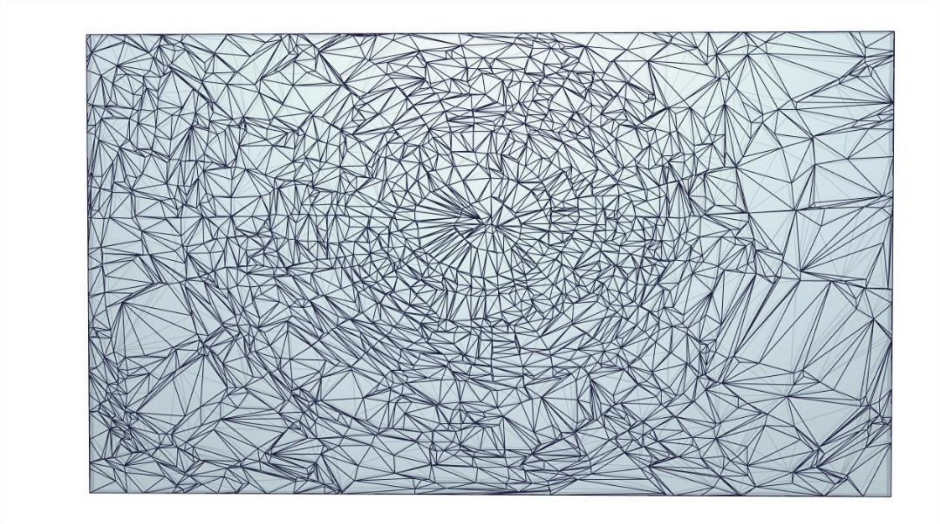
Je me suis alors penché sur le système de destruction DMM afin de réaliser le même effet de façon plus précise et plus réaliste. Après avoir créé et paramétré une surface avec suffisament de subdivision, j'ai créé une sphère animée pour venir entrer en collisison avec la vitre et venir la détruire. Le résultat obtenu est déjà beaucoup plus réaliste que précédemment. Néanmoins, je n'ai pas réussi à obtenir autre chose que des morceaux de débris relativement gros, contransatnt pour le coup avec les tests précédents.



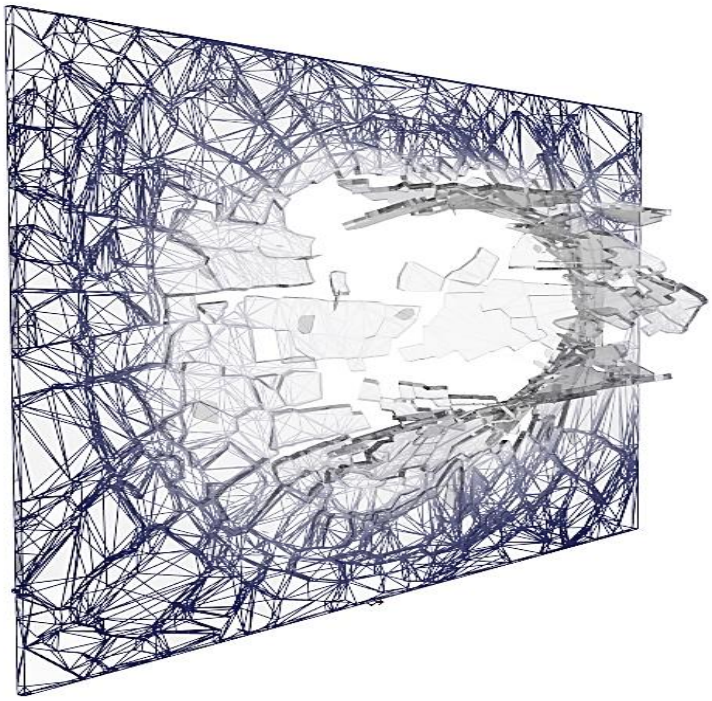
Je me suis alors demandé si les deux techniques ne pouvaient pas se compléter en étant utilisées en même temps. Les résultats obtenus furent déjà un brin plus convainquant et riche en terme de diversité de débris.



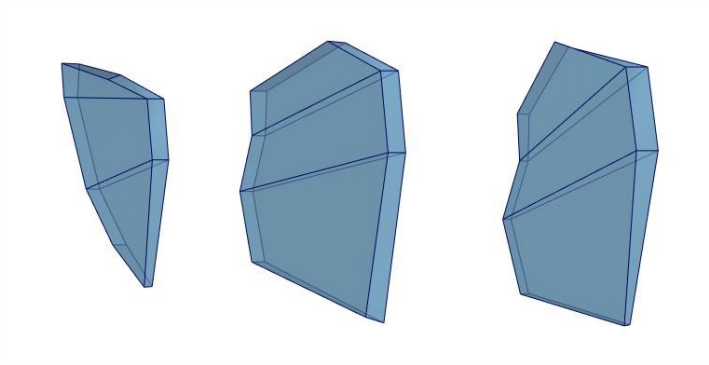
Néanmoins, je suis resté sur ma faim avec cette histoire de débris trop gros. C'est pourquoi j'ai tenté d'appliquer à ma vitre DMM un motif de destruction de meilleure qualité afin de la prédécouper de manière plus précise et plus réaliste en morceaux plus petits.



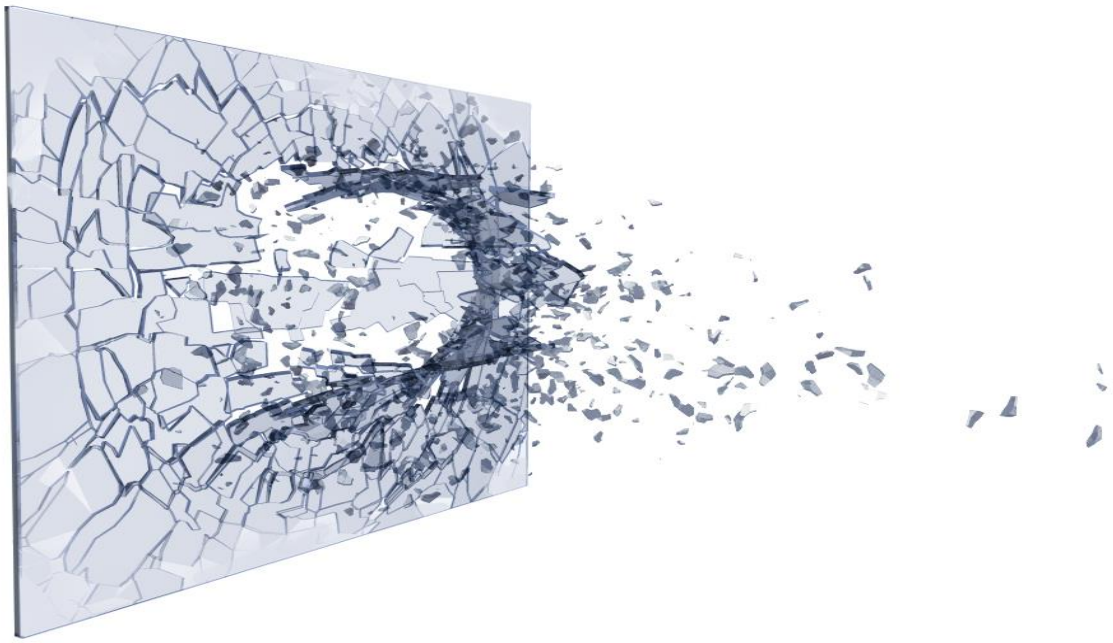
La destruction obtenue fut, effectivement, légèrement plus réaliste mais je n'arrivais toujours pas à obtenir de morceaux suffisamment fins, même en augmetant considérablement les paramètres de résolution de l'objet.



J’ai alors tenté d’utiliser le même subterfuge qu’auparavant, mais cette fois e, allant légèrement plus loin. J’ai donc créé un système de particules que j’ai instancé à trois morceaux de fragments différents pour créer une simulation visuellement plus variée et toujours moins uniforme. Puis, contrairement à mes premiers essais, j’ai émis ces particules, non pas depuis un émetteur rectangulaire animé, mais directement depuis l’intérieur des morceaux de verre fracturés par DMM.



Les résultats obtenus furent visuellement plus intéressant. Il m’a fallu néanmoins, faire attention à doser le nombre de particules émises, leur taille ainsi que la vitesse qu’elles héritent des fragments DMM.



2.2. Projets pyrotechniques simples

En guise de transition vers la troisième partie de ce mémoire, j'ai souhaité, dans les pages qui suivent, présenter étapes par étapes, deux petits projets de recherches sur la pyrotechnie de synthèses, que j'ai pu mener à bien du début à la fin.

2.2.1. La bougie

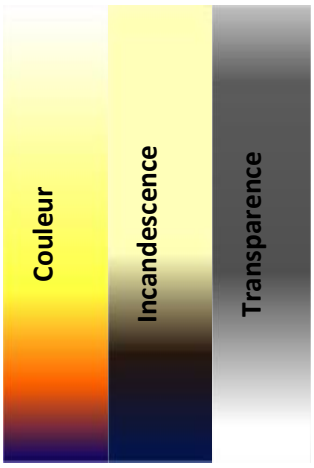
Comment reproduire la flamme vacillante d'une bougie avec un Soft Body ?

Créer la flamme

Première étape, une flamme est modélisée à partir d'un cube. La géométrie est ensuite convertie en Soft Body avec un *Goal* à 1. La valeur du Goal PP de chaque point de l'objet est ensuite modifiée à l'aide du *Paint Soft Body Weights Tool*. La base de la flamme doit moins bouger que la pointe, ses valeurs de *Goal PP* doivent alors être légèrement plus hautes. Un dégradé du gris clair au gris foncé est donc créé de bas en haut, soit des valeurs de *Goal PP* allant de 0,8 pour la base à 0,45 au le sommet.

Animer la flamme

Pour créer les perturbations dans la flamme, on applique aux particules du *Soft Body* une turbulence avec une faible fréquence. La géométrie est bel et bien perturbée seulement tous ses points semble agir indépendamment les uns des autres. Afin de créer une unité dans la flamme, des Springs sont appliqués à l'objet. En laissant simplement les valeurs par défaut, la flamme réagit correctement.



Créer la texture

Pour donner vie à notre flamme, il lui faut une bonne texture. On crée d'abord un mappage d'UV cylindrique sur l'objet pour que le haut de sa texture corresponde à son sommet et vice versa. Puis, dans un *Shader* de type *Lambert*, on connecte un dégradé pour chacun des paramètres de couleur, de transparence et d'incandescence, tel que l'image ci-contre.

Rendu et Compositing

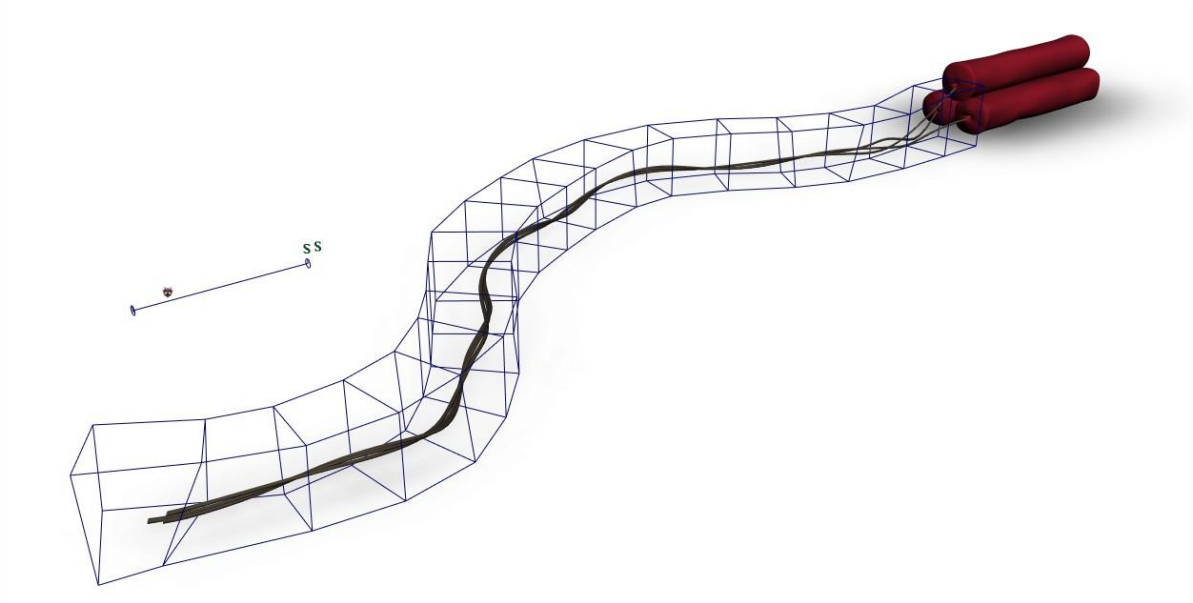
Pour le rendu seulement deux passes ont été nécessaires. Une passe contenant la mèche et la bougie avec les reflets de la flamme. Et une autre simplement avec la bougie sur fond noir. Au *compositing*, un effet de lueur diffuse a été ajouté à la flamme avant d'être superposé sur la bougie en mode de fusion *Screen*. Enfin, comme le veut la tradition, les effets de post-traitement habituels ont étés ajoutés ; aberration chromatique, grain, vignettage, tremblement de la caméra.

2.2.2. La dynamite

Comment donner vie à des étincelles le long d'un câble de dynamite ?

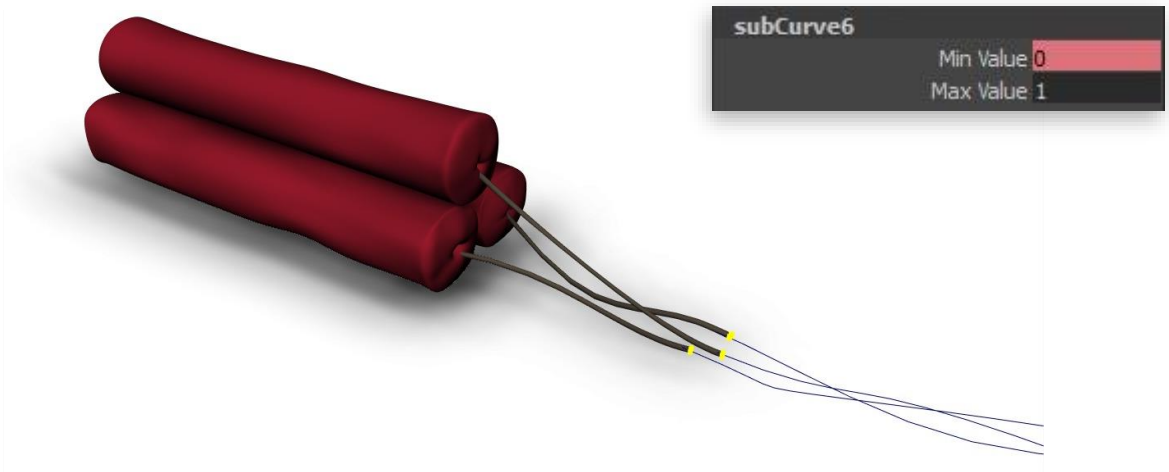
Modélisation de la mèche

Pour créer la mèche, un cercle est extrudé le long de trois courbes. Les cylindres obtenus sont ensuite entremêlés à l'aide d'un déformeur *Twist* et mise en place grâce à un *Lattice*.



Désintégration de la mèche

Pour animer la mèche qui disparait petit à petit, il suffit de jouer avec le paramètre *Min Value* de l'extrusion — attribut disponible lorsque l'on crée une extrusion de type partielle.



Émission des étincelles

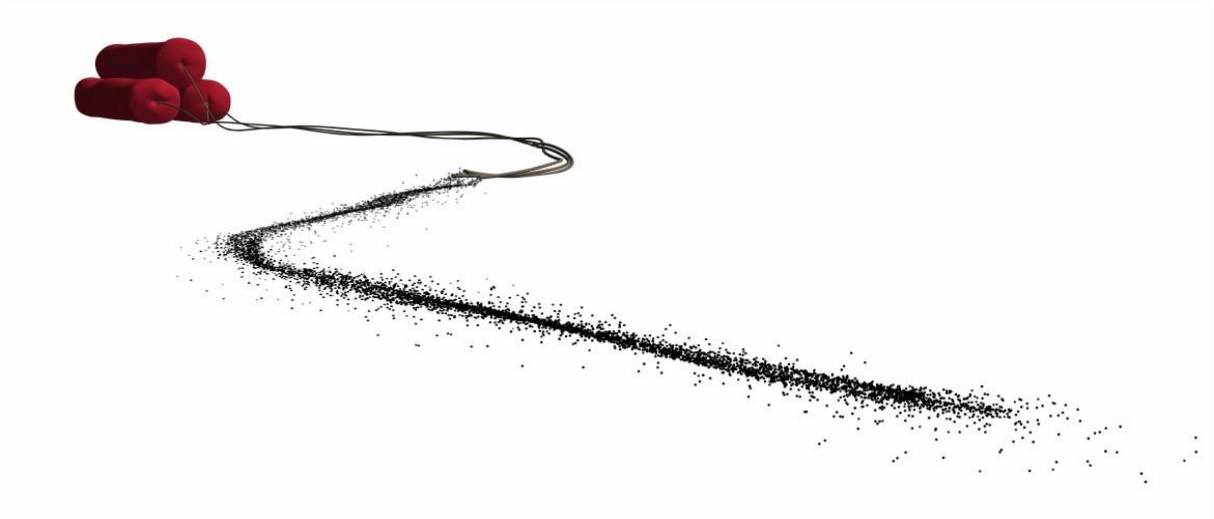
Pour créer les étincelles, des particules sont émises depuis les points de l'extrémité de la mèche. Pour les paramètres : un émetteur de type omnidirectionnel, une forte vitesse initiale, une faible durée de vie et un rendu de type *Streak*.

REMARQUE

Il est important d'utiliser le paramètre ***Min Value*** et non pas ***Max Value*** pour la désintégration de la mèche car de cette manière, les points qui se trouvent à l'extrémité de la mèche ne disparaissent jamais car ce sont les premiers à avoir été créés lors de l'extrusion. Autrement, il ne serait pas possible de les utiliser pour émettre des particules de manière continue.

Les résidus de mèche

Pour ajouter du détail, je souhaite faire apparaitre des résidus à longueur que la mèche se désintègre. Une solution consiste à émettre des particules depuis les mêmes points que pour les étincelles. Pour l'émission, un émetteur directionnel pointé vers le bas permet de répandre directement la traînée. Pour le sol, un plan pour lequel on active les collisions de particules — rebondissements nuls, friction au maximum.



Faire rougir les résidus

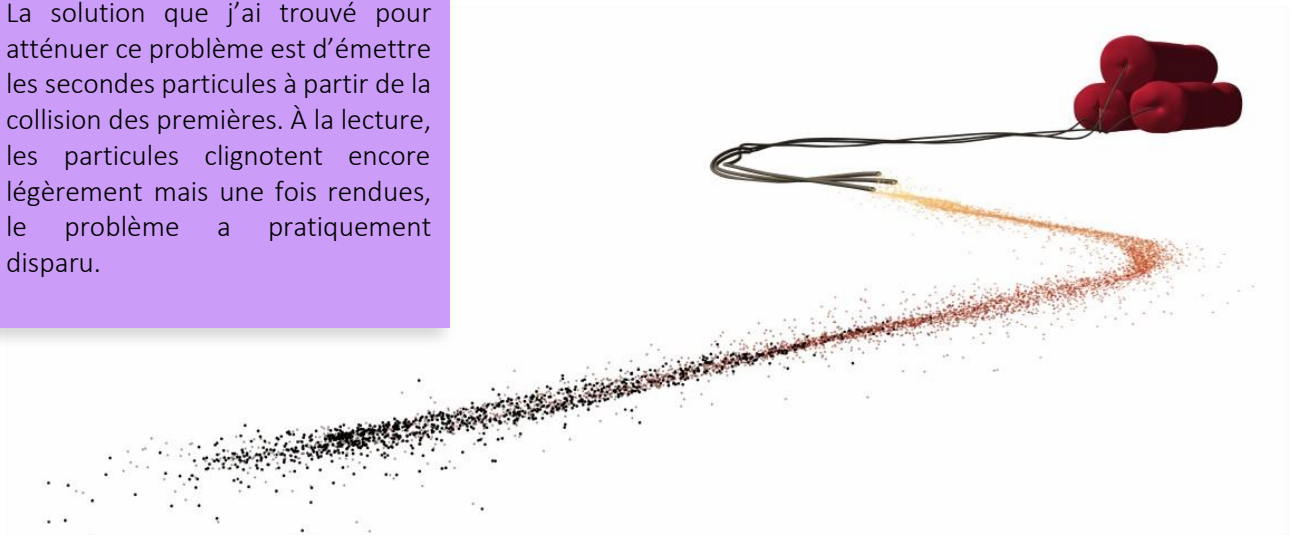
Lorsque les résidus sont émis ils sont d'abord chauds puis refroidissent. Je voudrais retranscrire cet effet au niveau de la couleur des particules — blanc à l'émission puis un dégradé jaune, orange, rouge pour arriver finalement au noir. Il est possible de changer la couleur des particules en fonction de leur durée de vie, le problème est qu'une fois la fin du dégradé atteint, les particules disparaissent.

Une solution qui fonctionne consiste à émettre une nouvelle particule noire à partir de chaque particule qui meurt. Pour ce faire, nous émettons des particules depuis celles existantes — *Emit From Object* — puis nous activons la possibilité de contrôler l'émission secondaire de chaque particule — *Per-Point Emission Rates*. Sur les premières particules, un nouvel attribut *RatePP* est apparu. Il permet de contrôler le nombre de particules secondaires émises par chaque première particule. Ainsi, en appliquant sur ce paramètre un dégradé basé sur l'attribut *LifespanPP*, nous pouvons contrôler le nombre de particules secondaires émises par chaque première particule en fonction de sa durée de vie. Il est donc possible d'en émettre uniquement à la mort de chaque particule.

PROBLEME RENCONTRE

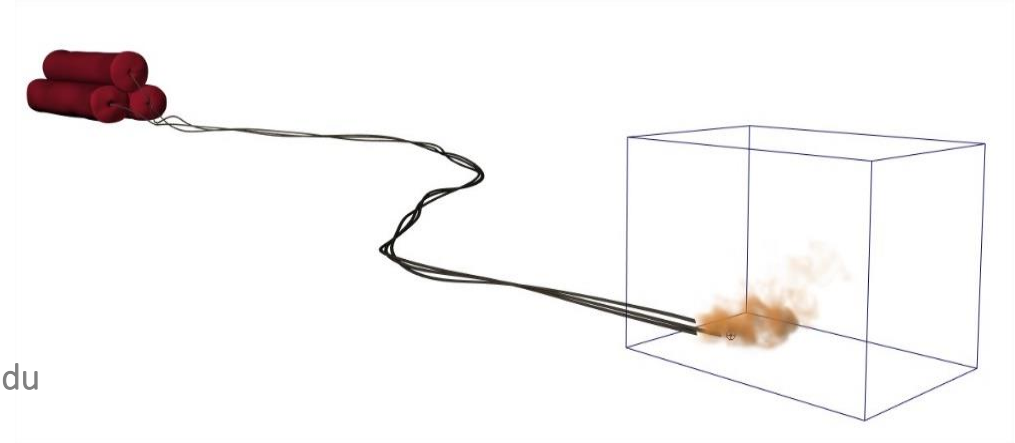
Afin que chaque particule émette à sa mort au moins une nouvelle particule pour la remplacer, j'ai été étrangement obligé d'augmenter l'émission secondaire à 100 particules par seconde. De ce faite, étant donné que les particules secondaires n'ont aucune vitesse initiale, il arrive qu'elles se superposent et créent des problèmes de clignotement au rendu.

La solution que j'ai trouvé pour atténuer ce problème est d'émettre les secondes particules à partir de la collision des premières. À la lecture, les particules clignotent encore légèrement mais une fois rendues, le problème a pratiquement disparu.



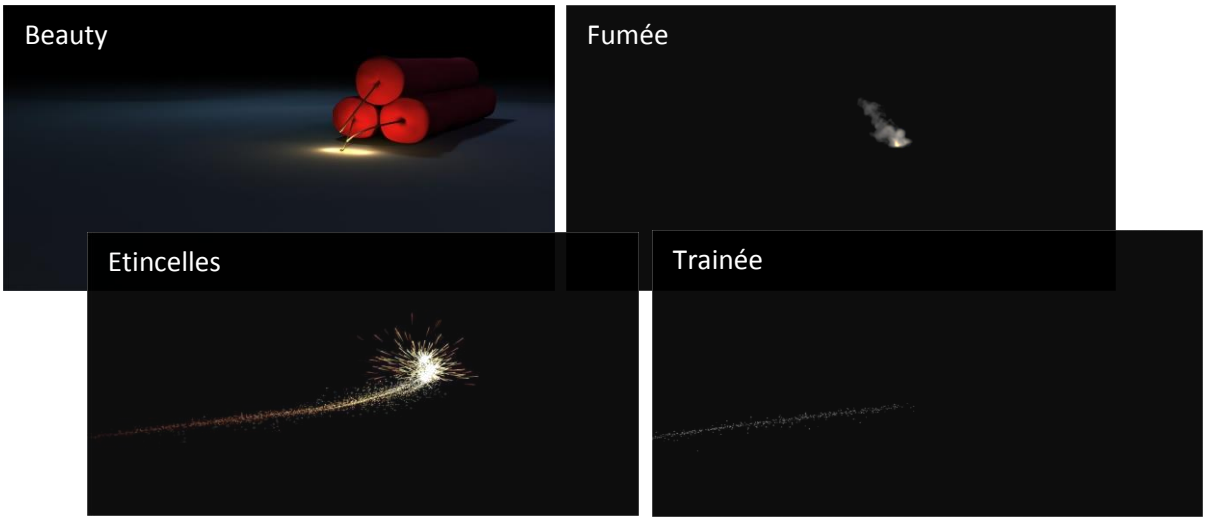
La fumée

Pour créer une petite fumée en plus des étincelles, un conteneur de fluide est parenté aux particules. Pour émettre le fluide, une nouvelle émission de particules est créée avec une vitesse importante dans la direction opposée à la mèche, et une durée de vie très courte. Au niveau du fluide, on augmente la vitesse héritée des particules — attribut *Inherit Velocity* — on ajoute ensuite de la turbulence et on règle la couleur et l'opacité de la simulation.



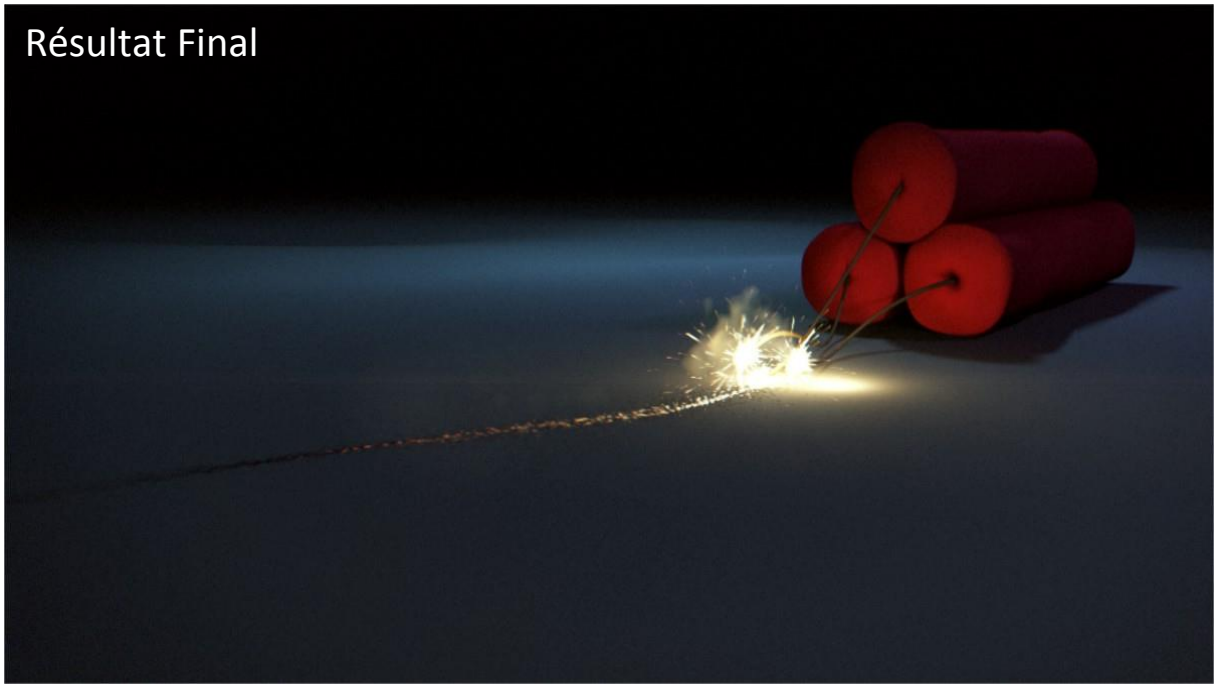
Le rendu

Pour le rendu, l'effet est séparé en quatre passes. Une passe principale contenant le rendu de la scène éclairée, sans particules ni fluides. Une passe de fumée avec le fluide éclairé. Une passe composée des étincelles et du début de la traînée de résidus. Et une passe avec la fin de la traînée à laquelle j'applique une couleur blanche pour qu'elle se détache du fond noir.



Le compositing

La passe de Beauty est notre base de travail. Par-dessus nous superposons la fumée et les étincelles en mode de fusion *Screen*. Puis la passe contenant la trainée est inversée et utilisée comme masque en luminance pour un effet d'assombrissement, ce qui crée une ligne de résidus noirs. Enfin quelques effets de post-traitement permettent d'inculquer à notre image les propriétés optiques habituelles ; aberration chromatique, grain, lumière parasite, vignettage, tremblement de la caméra.



PARTIE 3

Etude de cas : Matrix, explosion de l'hélicoptère

3.1. Présentation du projet

Entre particules, Fluides, *Soft* ou *Rigid Bodies*, force est de constater que ***Maya*** dispose d'un panel de fonctionnalités dynamiques très intéressant. Il nous serait donc possible, en théorie, de réaliser des effets complexes du début à la fin sans avoir à quitter le logiciel, ou du moins en ce qui concerne la phase de simulation dynamique et de rendu. J'ai donc choisi pour cette dernière partie de me consacrer à la réalisation d'une grande séquence issue d'un film existant afin d'essayer de reproduire des effets réalisés dans des conditions de production.

Pour le choix de la séquence, il m'a paru important de chercher un extrait suffisamment intéressant en termes de diversité d'effets spéciaux. J'ai finalement jeté mon dévolu sur un passage assez impressionnant de la fin du film ***Matrix*** ; le crash de l'hélicoptère (*Figure 4.1*). Selon moi, les plans qui composent cette séquence montrent une variété visuelle très intéressante en termes d'effets et de techniques à déployer pour leur réalisation. Destructions d'éléments de matériaux divers, onde de choc, fumée, feu, explosions, éclats de verre, débris multiples, autant d'éléments différents qu'il sera intéressant de faire cohabiter, autant en terme de simulation que de rendu et de composition visuelle.

Figure 4.1 Scène du crash de l'hélicoptère, *The Matrix,* Lana Wachowski, 1999.

3.2. Pré-production

Avant de me lancer dans le vif de la réalisation de cette séquence, la première étape est l'analyse visuelle de ma séquence référence. Pour mieux comprendre comment créer cet effet, je dois procéder à l'exercice inverse d'une production en partant du plan final pour remonter jusqu'aux méthodes et outils qui ont à priori été déployées pour sa conception. Selon moi, nous pouvons séparer la séquence en trois parties distinctes. L'onde de choc sur l'immeuble (*Figure 4.2*), l'éclatement des vitres (*Figure 4.3*) et l'explosion de l'hélicoptère (*Figure 4.4*). Viennent s'ajouter à cela une série d'éléments plus subtils tels que la légère fumée émanant de l'appareil, la déformation de l'hélice, la destruction de la façade au moment de la collision et les débris divers qui s'en suivent.

Figure 4.2 L'onde de choc, *The Matrix,* Lana Wachowski, 1999.

3.2.1. L'onde de choc

Premier effet, l'onde de choc (*Figure 4.2*). Au moment de la collision entre l'hélicoptère et le bâtiment, il se forme une vague en forme d'anneau qui s'étale en déformant la surface de l'immeuble avant de faire exploser toutes les vitres derrière elle. Avant de chercher les moyens pour déformer une surface géométrique, je me suis d'abord demandé s'il n'existait pas de solutions annexes permettant de contourner ce problème. Il existe tout d'abord la possibilité de donner l'illusion d'une déformation de

surface à l'aide de *Bump Map* ou de *Normal Map*, de simples textures que l'on connecte au matériel de notre géométrie afin de simuler du relief. Bien que ces techniques soient très efficace lorsqu'il s'agit de donner un peu de détail à des briques sur un mur par exemple, ce sont des effets qui restent limités car ils se contentent de simuler des rebonds de lumière sur la surface et ne créent réellement pas de déformation de la géométrie. Par conséquent, si l'on regardait notre objet latéralement, il n'y aurait aucune variation de la surface. Néanmoins, il existe une technique similaire appelé le *Displacement* qui va, lui, réellement modifier la position géométrique des points de la surface. Néanmoins, au vue de l'effet recherché, cette technique ne me semble pas appropriée pour deux raisons. La première est qu'il s'agit d'un effet que l'on utilise en général pour rajouter du détail sur des surfaces. Réaliser une déformation d'une telle ampleur n'aura sans doute pas la qualité escompté, surtout pour des gros plans. La seconde est d'ordre pratique. Étant donné qu'il s'agit d'une technique basée sur une texture il va donc falloir créer une séquence d'images animée à cet effet, afin que notre onde se déplace sur la géométrie. Pour la création de cette texture animée, il va donc falloir passer par un logiciel tierce sur lequel il faudra revenir sans cesse afin d'ajuster notre effet, ce qui rendra la tâche plutôt fastidieuse. Ainsi, selon moi, la déformation doit s'opérer non pas au niveau de la texture mais au niveau de la géométrie elle-même. Bien qu'il existe une multitude de techniques pour déformer un objet dans ***Maya***, la première qui me vient à l'esprit et qui me semble la plus appropriée dans cette situation, est l'utilisation d'un *Soft Body*. Comme nous avons pu le voir auparavant, un *Soft Body* permet de connecter des particules à une géométrie afin de la déformer en utilisant les propriétés dynamiques intrinsèques inhérentes aux particules dans Maya. Il s'agirait ici de créer une surface plane que l'on convertirait en *Soft Body* afin de venir la perturber pour créer un effet de vague similaire. La perturbation pouvant être effectuée soit à l'aide de champs de force soit à l'aide d'objets de collision.

3.2.2. L'explosion des vitres

L'effet résultant de cette onde est un éclatement des vitres du bâtiment (*Figure 4.3*). La projection des débris s'effectue en partant du centre vers le bord de l'immeuble sous forme d'une série d'anneaux de plus en plus grands.

Figure 4.3 **Explosion des vitres**, *The Matrix,* Lana Wachowski, 1999.

On peut, selon moi, distinguer trois couches à cet effet, ou plutôt trois niveaux de détail. Une partie d'abord composée de morceaux de verre plus ou moins épais, qui pourrait être réalisée à l'aide de plusieurs modélisations de débris de vitre que l'on utiliserait comme instance pour une émission de particules. Afin d'émettre ces particules, nous pourrions utiliser une géométrie primitive tel qu'un torus que l'on animerait afin qu'il grandisse au fil du temps, créant ainsi une diffusion circulaire. Deuxième niveau de détail, une simulation de fines particules censée représenter les tout petits fragments de verre qui s'échappent lors de l'explosion des vitres. Des fragments si petits qu'il serait inutile d'instancier les particules à des géométries car ils ressembleraient de toute façon à de simples points au rendu. Il s'agira alors de créer une simple émission de particules que l'on pourra animer en opacité pour simuler le scintillement du verre. Le torus émettant les morceaux épais pourra être réutilisé pour la génération de cette deuxième couche de détail. Enfin, la dernière partie de ces éclats de vitres est une couche que l'on pourrait qualifier de poussière de verre. Des particules si fines qu'elles s'apparentent d'avantage à de la fumée qu'à de véritables éclats. Étant

donné qu'une simulation de particules génère une multitude de points qui se comportent de façon dynamique mais indépendante, cette solution va être moins appropriée dans ce cas précis. Je me tournerais alors plutôt vers les Fluides *Maya* qui permettent de générer des simulations plus proches de phénomènes liquide ou gazeux. Pour la génération du fluide je pourrai utiliser la même géométrie d'émission que pour les particules avec une vitesse initiale dans la direction opposée au bâtiment.

Figure 4.4 Explosion de l'hélicoptère, *The Matrix,* Lana Wachowski, 1999.

3.2.3. L'explosion de l'hélicoptère

Dernier gros effet, l'explosion de l'hélicoptère (*Figure 4.4*). Une violente émanation de gaz dégageant une forte quantité de chaleur et de lumière qui laisse place à une épaisse fumée noire. L'explosion vient s'ajouter aux éclats de verre recouvrant le bâtiment. Elle retentit quelques secondes après le début de l'éclatement des vitres. Son déroulement se fait en deux étapes. L'hélicoptère semble exploser en deux endroits à quelques secondes d'intervalle. Pour réaliser cet effet, l'utilisation des Fluides *Maya* me semble être le choix le plus judicieux. Pour l'émission nous pourrions utiliser une combinaison entre émetteurs classique et particules. Au moment de l'explosion, on paramètrera une forte émission de chaleur et de carburant et tous les paramètres caractéristiques d'une explosion. Pour le rendu, il sera possible de jouer avec la texture d'incandescence pour donner plus de détail au Fluide.

3.2.4. Le rendu des éléments

En ce qui concerne le rendu, chaque effet devra évidemment faire l'objet d'une passe séparée, néanmoins il semble qu'il y ait quelques subtilités. En effet, étant donné que tous les éléments se trouvent dans le même espace 3D, à défaut d'interagir physiquement entre eux, il devrait au moins y avoir des interactions au niveau visuel. Par exemple, lorsque l'hélicoptère explose, les volutes de fumée s'élèvent dans le ciel et viennent progressivement masquer les débris de verre. Il faudra donc créer une passe de rendu qui permette de faire disparaitre les particules en fonction du fluide. Dans ce cas précis, une simple passe monochrome avec le fluide en noir et les particules en blanc devrait suffire à régler l'effet au *compositing*. Aussi, il pourra y avoir plusieurs passes pour chacun des effets comme par exemple des passes de lumière sur l'explosion afin de pouvoir réajuster l'éclairage après coup ou encore des passes de *réflexion* ou de *spéculaire* sur les particules instanciées pour avoir le contrôle sur leur intensité au *compositing*.

3.3. Production

Afin d'éviter de tomber dans la rédaction d'un tutoriel étape par étape, je présenterai de manière chronologique mon processus de réflexion et de création dans son intégralité. Je décrirai mes étapes de création au fur et à mesure, sans omettre les problèmes rencontrés, les solutions mises en place pour les contourner, les impasses, les changements de cap ou les erreurs inhérentes au logiciel ou à la technique utilisée.

3.3.1. L'onde de choc

a. Perturbation d'un Soft Body à l'aide d'un champ de force

Pour débuter le travail de cet effet je décide de me lancer tout d'abord sur l'onde de choc. Je crée alors un plan avec suffisamment de subdivisions pour que sa déformation soit la plus précise possible et de qualité suffisante mais sans pour autant que ce soit trop coûteux en temps de calcul. Je convertis la surface en *Soft Body* en conservant la géométrie originale et en l'utilisant comme *Goal* pour le *Soft Body*. Je crée ensuite un champ de force de type *Air Field* auquel je désactive le paramètre d'amplitude et d'atténuation. Le voilà inactif face à mes particules. J'augmente alors son paramètre d'héritage de vitesse afin qu'il transmette sa vitesse aux éléments qui l'entourent et sur lesquels il agit. Ainsi, après avoir vérifié qu'il soit bien connecté aux particules de mon *Soft Body*, je crée une animation sur le champ de force pour qu'il passe au travers du plan et je regarde le résultat (***Figure 4.5***). L'effet fonctionne, les particules sont déplacées au passage de l'*Air Field* et la surface est déformée en conséquence. Par défaut, le champ de force ne dispose d'aucun volume.

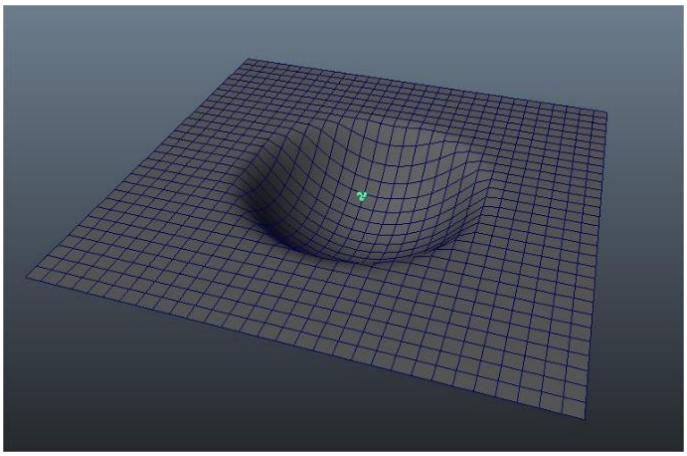


Figure 4.5 Déformation d'un Soft Body au passage d'un Air Field.

Son rayon d'émission est alors dicté par son paramètre de distance maximum qui se règle en nombre d'unités. Afin que je puisse régler l'effet d'une manière plus visuelle, j'active le volume en forme de sphère et l'agrandit à une taille raisonnable. L'effet est alors localisé dans la zone de ce volume. La première étape de l'effet semble être prête, reste alors à créer l'onde de choc résultante.

b. Propagation de l'onde grâce à l'outil Springs

Etant donné que mon *Soft Body* est composé de particules qui sont chacune indépendantes, il est normal que lorsque je les perturbe dans un rayon défini, les particules qui sont en dehors de ce rayon ne soient pas affectées. Néanmoins, ce que je souhaiterais ici c'est faire réagir la totalité de ma surface à l'impact qui lui est donnée à son centre, comme s'il s'agissait de la surface d'une étendue d'eau. Pour réaliser cet effet, j'applique à mes particules l'outil *Springs* afin de les relier entre elles par des sortes de ressorts virtuels qui vont sans cesse tenter de les faire revenir à la distance qui les séparaient initialement, et unifier ainsi la surface lors de sa déformation. A la lecture, le comportement des particules semble être sensiblement le même. Même après avoir ajusté les paramètres liés à la rigidité du ressort et à la vitesse de perte d'énergie, la déformation de mon plan reste localisée à son centre et ne se propage pas jusqu'à ses extrémités (*Figure 4.6*).

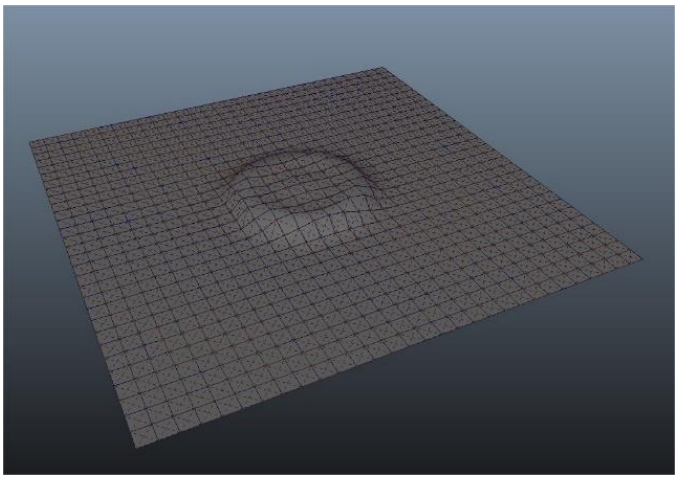


Figure 4.6 Déformation d'un Soft Body avec un Springs au passage d'un Air Field.

c. Déformation du Soft Body par un objet de collision

Je décide alors de tenter une autre technique consistant à faire entrer en collision le *Soft Body* avec un objet en espérant que la réaction soit plus marquée. Je crée une sphère, la convertit en *Rigid Body*, lui applique une gravité et permet son interaction avec les particules. Je lance l'animation. Au moment de la collision entre les deux objets, la sphère est immédiatement éjectée dans le sens opposé au contact et ne fait que faire frémir le centre de ma surface. J'augmente alors la masse de ma sphère pour qu'elle ait plus d'impact. Le plan semble effectivement réagir d'avantage mais revient trop vite à son état initial sans aucun effet de vague semblable à ce que je recherche. Je décide alors de baisser le paramètre *Goal Weight* de mes particules afin que le *Soft Body* mette plus de temps avant de retrouver sa forme initiale. Le résultat est plus concluant (***Figure 4.7***). A la lecture, la sphère entre en collision avec le plan et crée une vague beaucoup plus ample et qui déforme le plan d'une façon analogue à la surface d'un liquide. La sphère est toujours également éjectée à sa collision mais c'est plutôt en notre faveur dans ce cas puisqu'une fois la vague activée, la sphère ne nous sert plus à rien. On peut alors activer ses clés d'animation pour qu'elle passe en *Rigid Body* passif au moment où elle se retrouve en l'air afin qu'elle ne bouge plus et n'entre plus en collision avec notre surface.

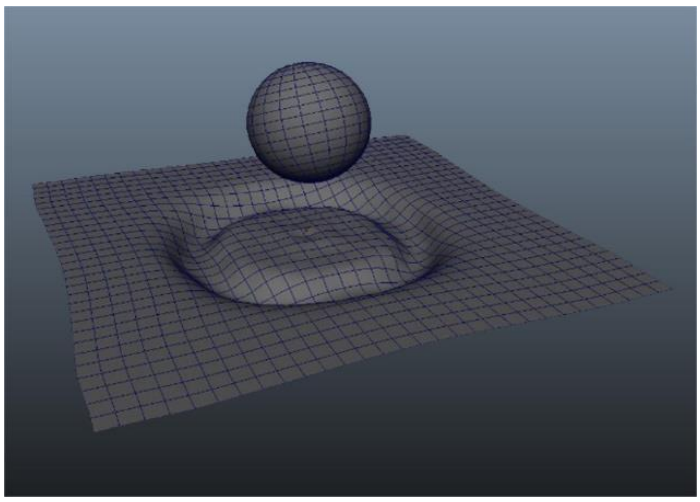


Figure 4.7 Déformation d'un Soft Body au contact d'un Rigid Body

d. Premières analyses du résultat

A présent notre surface réagit correctement, néanmoins je dénote plusieurs problèmes. Premièrement les bords de notre surface réagissent au moment de la collision alors qu'ils devraient rester fixes avant que la vague ne vienne les perturber. Deuxièmement, l'effet de vague crée non seulement des vagues en forme de cercle similaire à l'effet recherché, mais aussi des tumultes un peu aléatoires venant perturber l'ensemble de la surface (*Figure 4.8*). C'est un problème car, si l'on se réfère à l'effet de *Matrix*, la vague que l'on souhaite obtenir doit être beaucoup plus lisse.

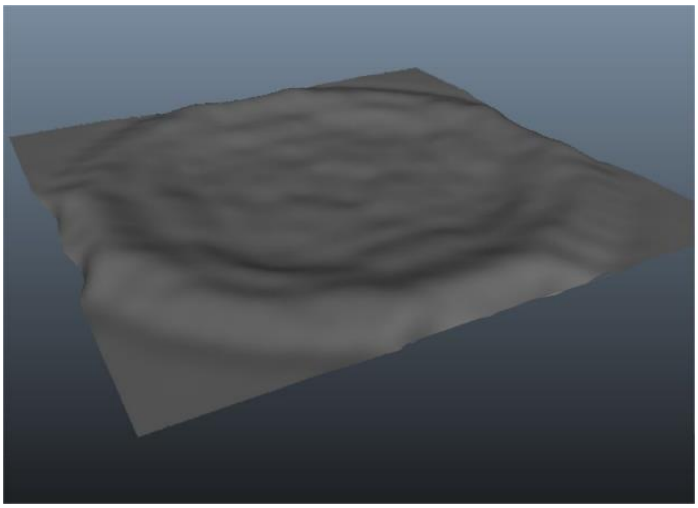


Figure 4.8 Surface bruitée d'un Soft Body après le contact avec un Rigid Body

Enfin, je n'ai aucun contrôle sur la vitesse de propagation de la vague. Si je souhaite qu'elle se déplace plus vite, je vais soit devoir baisser le *Goal* de mon *Soft Body* soit augmenter la masse de l'objet qui rentre en collision — ou la magnitude de la gravité — afin qu'il tombe plus vite. Le problème est qu'en changeant la vitesse de la vague, je vais aussi modifier la hauteur de celle-ci ainsi que les perturbations parasites dont je voudrais me débarrasser. Il faudrait donc trouver un moyen de contrôler indépendamment la vitesse à laquelle se déplace la vague et sa taille.

e. Perturbation du Soft Body à l'aide d'un torus animé

Je pense alors à créer non pas une sphère mais un torus qui va entrer en collision avec la surface et dont on animera le rayon pour qu'il grandisse à la vitesse qui nous intéresse et vienne perturber le *Soft Body* à la bonne amplitude. Je crée alors la primitive, l'anime et active ses collisions avec les particules de mon *Soft Body*. A la lecture, le résultat semble à première vue satisfaisant. Néanmoins, lorsque l'on y regarde de plus près, on se rend compte que la surface réagit de manière étrange au niveau des contours de l'onde de choc. Il se forme, en effet, de petits artefacts ressemblants aux plis d'un tissu qui semble inévitable si l'on veut garder ce niveau de détail et cette amplitude de déformation.

f. Réflexion et changement de stratégie

Je me dis alors que j'ai sans doute dû prendre la mauvaise direction. Afin d'être le plus précis possible et d'obtenir réellement l'effet que je cherche, je ne dois pas venir perturber mon *Soft Body* à l'aide d'un objet tiers, je dois penser en terme de *Goal*. Étant donné que les particules de mon *Soft Body* utilisent comme *Goal* la surface d'origine, si je déforme cette surface, mon *Soft Body* va se déformer en conséquence puis mettre un certain temps avant de revenir à sa forme initiale. Il faut donc trouver un moyen de déformer mon plan original de façon à faire émerger un anneau se propageant du centre aux contours de ma surface. Après quelques rapides recherches, je trouve un effet intéressant en appliquant à mon plan un déformeur non linéaire, le *Wave Deformer*. Il s'agit d'un outil qui permet de déformer une géométrie en créant justement une série d'ondes circulaires dont on va pouvoir gérer la fréquence, l'amplitude et la phase. Tous ces paramètres ont évidemment la possibilité d'être animés, il serait donc a priori possible de s'en servir pour créer un effet d'onde qui se propage sur notre surface.

g. Création d'une onde sur un plan à l'aide d'un déformeur

Me voilà à présent reparti à zéro. Je crée donc un plan avec suffisamment de subdivisions pour que sa déformation soit la plus précise possible et de qualité suffisante mais sans pour autant que ce soit trop coûteux en temps de calcul. Néanmoins, je ne convertis pas tout de suite cette surface en *Soft Body*, je lui applique, en revanche, un déformeur de type *Wave* (***Figure 4.9***). Je joue alors avec les paramètres de l'outil pour tenter de régler l'effet. Je commence par baisser la taille de la vague pour que l'anneau soit relativement mince puis j'augmente l'amplitude afin que l'effet soit suffisamment marqué. Pour le déplacement de l'onde, je teste d'abord une animation sur la taille du déformeur associée à une animation sur le paramètre lié à la taille de la vague, afin que l'anneau reste à peu près à la même épaisseur lorsqu'il grandit. Le résultat est peu concluant. Le centre de l'onde est un peu étrange et il est impossible d'obtenir une onde suffisamment marquée et qui se détache des autres vagues. Je tente alors d'animer uniquement les rayons minimum et maximum du déformeur afin de créer un anneau qui parte du centre et se propage vers l'extérieur de la surface. Le résultat fonctionne mieux (***Figure 4.10***). Le seul problème est que la hauteur de ma surface avant le passage de

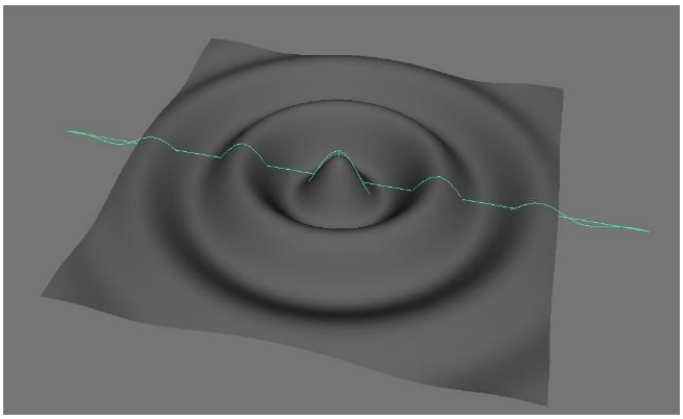


Figure 4.9 Plan perturbé par un déformeur de type Wave.

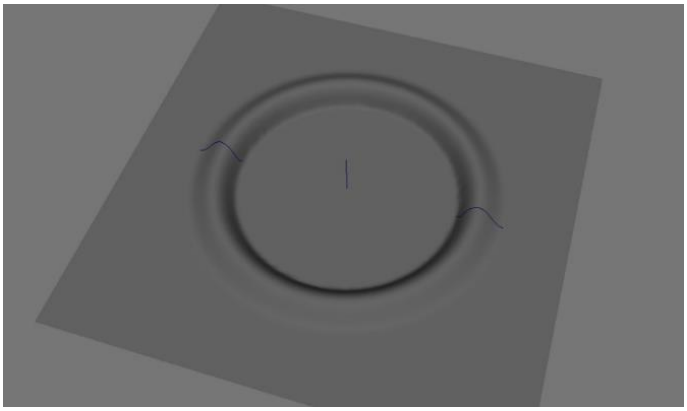


Figure 4.10 Déformeur paramétré pour déformer le plan selon un anneau.

l'anneau n'est pas la même qu'après son passage. Afin de régler ce détail, je modifie légèrement le paramètre d'offset du déformeur. Le tour est joué !

h. Conversion du plan en Soft Body pour simuler des remous

L'anneau de déformation se propage, à présent, correctement le long de ma surface. Il est donc temps de la convertir en *Soft Body* afin de simuler l'onde additionnelle qui devrait venir perturber la surface juste après le passage de l'anneau de déformation. Je crée donc un *Soft Body* à partir de mon plan déformé en prenant bien soin d'utiliser la géométrie originale comme *Goal*. Je laisse les paramètres par défaut et je lance l'animation ; le résultat fonctionne (*Figure 4.11*). Je dénote néanmoins un effet de remous un peu trop marqué après le passage de l'anneau. Je décide donc d'augmenter légèrement le paramètre de *Goal Weight* du *Soft Body* afin que cet effet soit moins important.

Figure 4.11 Onde sur Soft Body générée par un déformeur en forme d'anneau.

3.3.2. Les éclats de vitres — Partie 1

Pour la mise en place des débris de verre issus de l'explosion des vitres, j'ai commis plusieurs erreurs. J'ai, en effet, voulu aller un peu trop vite en besogne, ce qui a finalement produit l'effet inverse et m'a fait perdre du temps. Néanmoins, mes différents essais, ratages et retours en arrière m'ont tout de même permis de prendre conscience des avantages et inconvénients des différentes techniques testées.

a. Création d'un émetteur de particules en torus

Ma première idée est de me servir d'un émetteur de type volume avec une forme de torus. Après l'avoir créé, je règle ses paramètres de vitesse d'émission afin que les particules soient émises assez rapidement et de façon perpendiculaire au volume — j'utilise pour ça les attributs *Direction* et *Directional Speed*. Pour créer une simulation un peu moins uniforme, j'augmente les paramètres d'aléatoire au niveau de la vitesse et de la direction (***Figure 4.12***) — *Speed Random* et *Random Direction*. Je souhaite ensuite créer une animation du rayon de mon torus afin qu'il grandisse au fil du temps. Néanmoins, je constate que dans les paramètres de l'émetteur j'ai accès à un attribut permettant de régler l'épaisseur — l'attribut *Section Radius* — mais aucun pour la taille de mon rayon. Pour animer la taille du torus, je crée donc une animation sur son *Transform Node*. Le problème est qu'en augmentant sa taille, j'augmente par la même occasion son épaisseur. Il va donc falloir animer l'attribut *Section Radius* de façon inversement proportionnelle à la taille, pour que l'épaisseur du torus reste relativement fixe. L'inconvénient est qu'à chaque fois que je voudrai changer l'animation du rayon, je devrai aussi changer celle de l'épaisseur. Je décide donc de créer une expression sur l'attribut *Section Radius* pour que sa valeur soit toujours l'inverse de l'attribut *Scale*. Après quelques essais, j'obtiens finalement le code suivant : "*sectionRadius = (1 / scaleX) / 7 + 0.005 ;*". A présent, peu importe la valeur de la taille, l'épaisseur du volume reste toujours sensiblement la même.

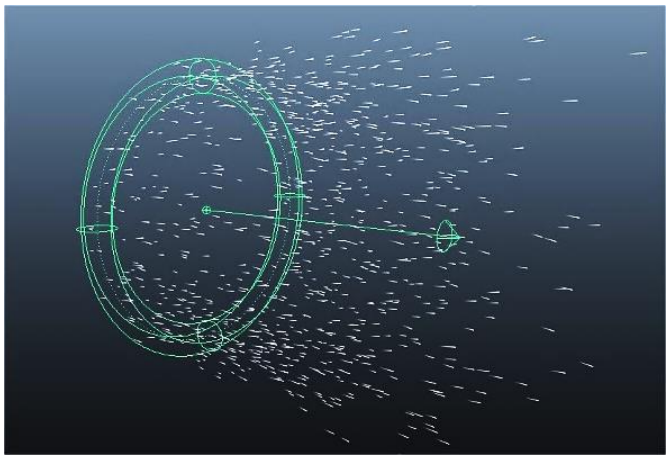


Figure 4.12 Particules générées depuis un émetteur de type volume en torus.

b. Analyse des problèmes

Le problème avec cette technique concerne le paramétrage des différentes couches de débris. En effet, bien qu'il soit possible de connecter plusieurs *Particle Objects* sur un seul émetteur, il sera ensuite impossible de conférer des paramètres différents à chaque couche de simulation. Autrement dit, si je n'ai qu'un seul émetteur, je n'aurai qu'un seul paramétrage. Entre mes différentes simulations de débris de verre, je ne pourrai alors pas faire varier la direction, la vitesse d'émission, le nombre de particules, etc. Il serait néanmoins possible d'utiliser des champs de force associés aux différentes particules afin de créer des variations, mais le réglage de l'effet pourrait rapidement devenir trop lourd et assez confus. Aussi, la poussière de verre, censée être simulée par des fluides, ne pourra être émise autrement que par des particules, ce qui limite les possibilités. Enfin, lorsque l'on utilise un *Volume Emitter*, il est impossible de connecter l'émission des particules à une texture pour casser l'uniformité de la simulation.

c. Création d'un anneau polygonal pour l'émission des particules

Ma deuxième idée pour l'émission des particules est de me servir d'un anneau polygonal que j'aurais extrudé à l'aide d'une ligne le long d'un cercle. De cette façon, il me sera possible de connecter une texture à l'émission, d'émettre directement un fluide depuis cette surface, et d'avoir des paramètres d'émission différents pour chaque simulation de particules — car chaque système de particules, même s'il sera émis depuis la même surface, possédera son propre émetteur. Je crée alors deux courbes ; un cercle et une ligne.

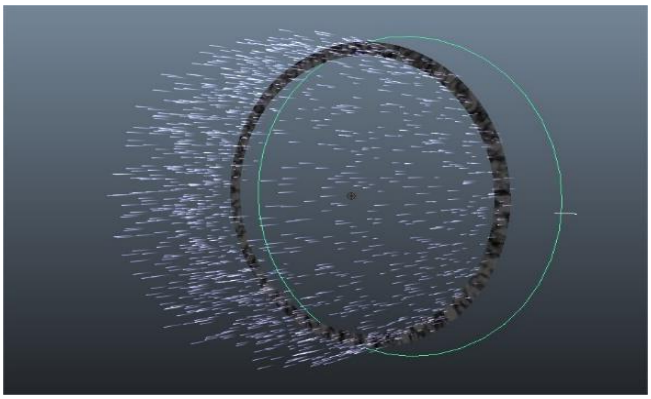


Figure 4.13 Particules émises depuis un polygone résultant d'une extrusion de courbes.

J'extrude la ligne le long de mon cercle en réglant correctement les paramètres d'orientation au moment de la création. J'ai à présent une surface polygonale en forme d'anneau toujours contrôlée par mes deux courbes. La taille de la ligne est liée à l'épaisseur de l'anneau et le rayon du cercle est lié au rayon de l'anneau. Ainsi, si je prends soin de ne pas supprimer l'historique de construction, il m'est possible de modifier le rayon et l'épaisseur de l'anneau à tout moment en modifiant les courbes de construction. J'utilise alors le cercle pour animer le rayon de ma surface d'émission et je crée une simulation de particules en choisissant *Emit From Object* (***Figure 4.13***). Puis, afin que mes débris de verre apparaissent par salves, j'anime le paramètre *Rate*, qui gère le nombre de particules générées (***Figure 4.14***).

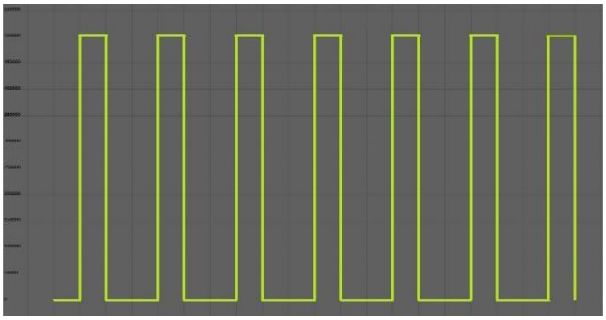


Figure 4.14 Animation par intermittences du paramètre Rate d'un système de particules.

d. Analyse des problèmes

Le problème de cette technique est qu'il est très difficile de gérer correctement la position des émissions de débris et leur timing car les deux sont étroitement liés. En effet, si je veux changer, par exemple, la vitesse à laquelle sont émis les différents cercles de particules, je vais devoir modifier l'animation du paramètre *Rate* de l'émetteur, ce qui va changer la taille des différents anneaux d'émission. Je devrai alors modifier également l'animation liée au rayon du cercle. Autrement dit, une fois que la vitesse à laquelle apparaissent les différentes explosions me convient, la position — rayon du cercle — ne correspond plus, et vice versa. Ainsi, afin d'avoir un meilleur contrôle de l'effet, je dois trouver une méthode qui me permette de gérer séparément la taille des différents anneaux d'émission et la vitesse de génération des particules.

e. Recherche d'une nouvelle méthode

La solution la plus basique serait de créer autant d'anneaux qu'il y a d'explosions, de faire un émetteur distinct à partir de toutes ces géométries puis d'animer l'émission des particules de tous ces émetteurs. Cette méthode semble néanmoins assez fastidieuse à mettre en place et risque de créer des confusions lors du réglage des paramètres. Une solution qui ne nécessiterait qu'un seul émetteur serait de créer une surface d'émission plane avec une texture d'émission comportant une animation avec les différents anneaux d'émission animés. La difficulté serait alors dans la réalisation de cette texture. Car si elle est faite dans un logiciel de motion design tel qu'*After Effects* et ensuite importé dans Maya, nous perdons l'interactivité. Par conséquent, si la vitesse ou le rayon de l'anneau ne nous convient pas il faudra retourner sous *After Effects*, modifier l'animation et relancer un calcul, ce qui est aussi une source de perte de productivité. Il serait donc intéressant de trouver le moyen de réaliser cette texture dans *Maya*.

f. Création d'une texture d'émission à l'aide de dégradés

Me vient alors à l'idée de partir d'un dégradé de type circulaire pour créer les différents anneaux puis de les faire apparaitre et disparaitre successivement en animant la couleur. On aurait alors le contrôle sur la vitesse d'émission et sur le diamètre des anneaux. Je crée donc le dégradé et l'applique sur une géométrie plane pour avoir un retour visuel (*Figure 4.15*). Je prends soudain conscience que l'animation de toutes les

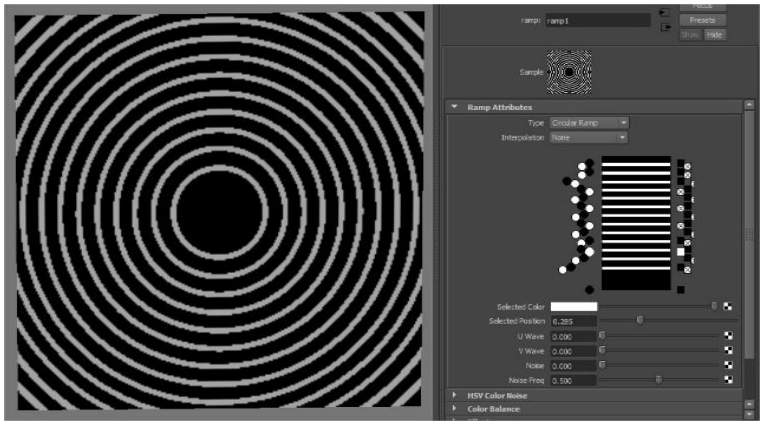


Figure 4.15 Texture d'anneaux noir et blanc créée à l'aide d'un dégradé.

couleurs risque d'être rapidement confuse et fastidieuse vu le nombre d'attributs de couleur concernés. Je décide alors de créer un deuxième dégradé, plus simple, comprenant un seul anneau que je pourrai animer plus facilement, afin de m'en servir comme masque pour le dégradé précédent. Ainsi, mon premier dégradé me sert à définir la taille des différents anneaux d'émission et le second me sert à régler la vitesse d'apparition de toutes ces émissions successives. Enfin, je termine ma texture en ajoutant du Noise dans le nœud de placement de texture connecté au second dégradé, afin de casser le motif d'émission (*Figure 4.16*). Je peux alors passer au réglage des particules.

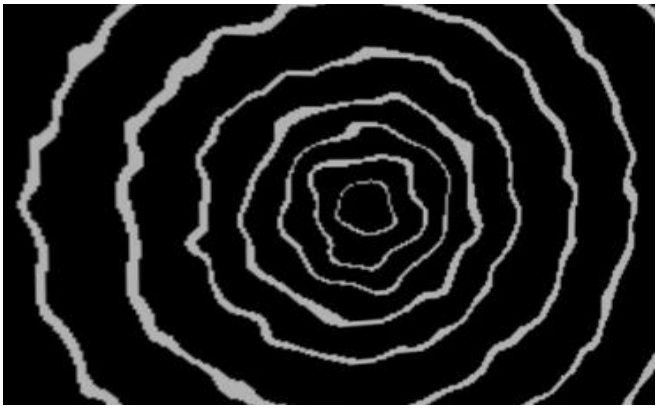


Figure 4.16 Texture d'anneaux noir et blanc créée avec un dégradé et perturbé par un Noise.

g. Création de la première couche de débris (morceaux de verre)

Je crée alors un système de particule que j'émets depuis un plan et je connecte au paramètre *Texture Rate* la texture d'émission animée réalisée précédemment. Par défaut le système n'émet pas assez de particules. Afin d'en obtenir une quantité satisfaisante, je dois monter à un million le nombre de particules par seconde. A présent, comme je dois faire éclater les vitres en une série de petits morceaux de verre, je modélise cinq fragments différents et m'en sers comme instances pour mes particules. De cette façon chaque particule se retrouve remplacée aléatoirement par l'une de ces géométries. Afin que tous mes débris ne soient pas tous de la même taille, je crée un attribut personnalisé que j'appelle "ScalePP" et auquel j'assigne un nombre aléatoire à l'aide de l'expression suivante : " *scalePP = rand (0.01 , 0.55);*". Je connecte enfin cet attribut au paramètre *Scale* dans le menu des instances. Après avoir effectué le même type d'opération à la rotation, les morceaux de verre générés sont tous d'une taille et

une orientation différente. Reste à présent à travailler le mouvement des particules.

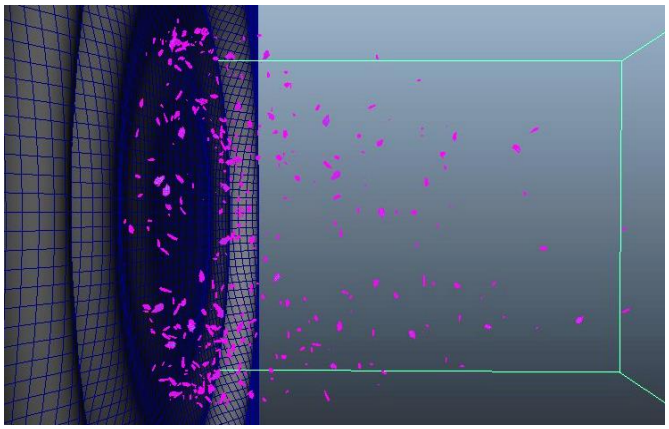


Figure 4.17 Gros fragments de verre créés à l'aide d'instances de particules émises depuis un plan.

Pour que l'impulsion de départ soit suffisamment marquée, j'augmente considérablement la vitesse initiale le long de la normale de la surface d'émission. Je fais ensuite perdre de la vitesse aux particules à l'aide d'un champ de force *Drag* et les attire légèrement vers le bas avec une gravité. Enfin, pour casser l'uniformité de la simulation, j'applique une turbulence aux particules (*Figure 4.17*).

h. Création de la deuxième couche de débris (Fines particules)

Je passe à présent à l'émission des particules de verre. Toujours à partir de la même surface, je crée un nouveau système de particule. Après quelques réglages de la simulation, je réalise qu'afin d'avoir un nombre raisonnable de particules, je dois pousser l'émission à dix millions de particules par secondes. Le problème est que le temps de calcul devient alors considérablement long. En descendant le nombre de particules émises par seconde autour d'une valeur de cent mille, soit cent fois moins, la vitesse de simulation devient alors beaucoup plus raisonnable. Néanmoins, un nouveau problème fait surface, le nombre insuffisant de particules. Je tente alors de créer une émission de particules secondaires à partir de ces particules — autrement dit j'émets des particules depuis mes particules existantes. Le résultat est alors surprenant. En effet, en réglant le second émetteur à deux-cent particules par secondes, je me retrouve avec une émission de vingt millions de particules par secondes, soit deux fois plus de particules qu'auparavant, pour une vitesse de calcul proche du temps réel — environ vingt-deux images par secondes. Je règle alors le mouvement des particules, à la manière de la simulation de débris précédente — vitesse initiale importante, Drag,

turbulence, gravité, etc. — en tenant compte du fait qu'il s'agit, ici, d'une couche de débris plus petite, donc plus légère — je n'utilise donc pas les mêmes champs de force qu'auparavant (*Figure 4.15*). Enfin, afin que mes particules clignotent comme le feraient des particules de verre, j'applique à chaque images une valeur aléatoire sur l'opacité de chaque particule: "*opacityPP = rand(0.01 , 0.3);*".

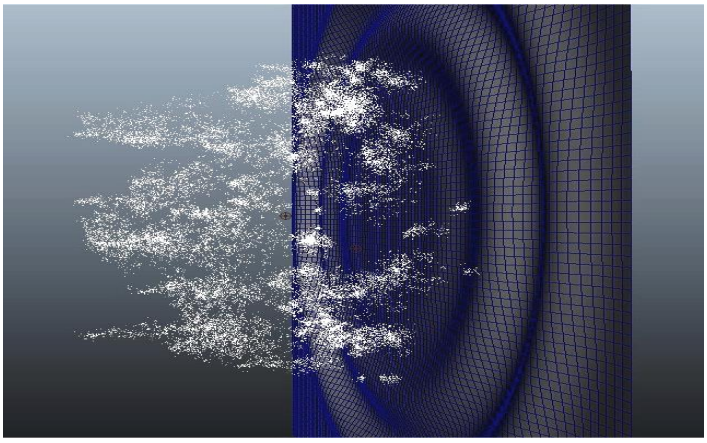


Figure 4.18 Débris de verre fins réalisés à l'aide de simples particules émises depuis un plan.

i. Problèmes rencontrés et solutions mises en place

En laissant tourner la simulation, je remarque certains problèmes. Tout d'abord, je me rends compte que certaines particules se retrouvent émises de l'autre côté de ma surface — vers l'intérieur de la façade de l'immeuble. Bien que cela ne risque pas d'être un problème lors du rendu — puisqu'elles seront cachés derrière un plan —, générer plus de particules que nécessaire peut ralentir les calculs, surtout lorsque l'on parle de millions de particules. Ainsi, afin de supprimer les particules qui se retrouvent de l'autre côté de ma surface, je crée une expression qui change la durée de vie à zéro pour toutes les particules qui franchissent une certaine position dans l'espace et ce à chaque seconde: "*vector $pos =position; if ($pos.z < -1) lifespanPP = 0;*".

Deuxième problème, certaines particules se retrouvent bloquées à l'endroit où elles semblent avoir été émises. Bien qu'elles tremblent légèrement, leur vitesse est assez basse contrairement aux autres particules. Par conséquent, il nous sera possible de les isoler à partir de ce paramètre. Afin de vérifier si ma théorie fonctionne, je crée une expression sur les particules qui change leur couleur en fonction de leur vitesse. Au-

```
Expression:
float $speed = mag(velocity);
if ($speed < 5)
rgbPP = <<1,0,0>>;
```

Figure 4.19 Expression qui affecte une couleur des particules en fonction de leur vitesse.

```
Expression:
float $speed = mag(velocity);
if ($speed < 5 && $pos.z< 13.5)
lifespanPP = 0;
```

Figure 4.20 Expression qui supprime des particules en fonction de leur vitesse et position.

dessous d'une certaine vitesse, les particules deviennent rouge (*Figure 4.19*). Je me rends alors compte que les particules stagnantes changent bien de couleur mais d'autres particules, ayant perdu leur vitesse à cause du *Drag Field*, deviennent rouge à leur tour. Je décide alors de contraindre ce changement de couleur aux particules se trouvant proche de la surface d'émission. Le résultat est plus concluant. J'adapte alors mon expression pour qu'au lieu de changer de couleur, les particules concernées disparaissent (*Figure 4.20*).

j. Problèmes lors de la mise en place de la dernière couche

Pour la dernière couche de débris, je souhaite créer une poussière de verre à l'aide d'une simulation de fluide qui réagit d'une façon semblable aux deux systèmes précédents. Afin de récupérer un mouvement similaire, je crée une troisième simulation de particules que je modifie légèrement puis que j'utilise comme émetteur pour ma simulation de Fluides. Pour mon Fluide, je crée une simulation de poussière blanche légèrement diffuse et affectée par la gravité pour qu'elles retombent une fois émise. J'active également le paramètre permettant aux particules de transmettre leur mouvement au fluide lors de sa

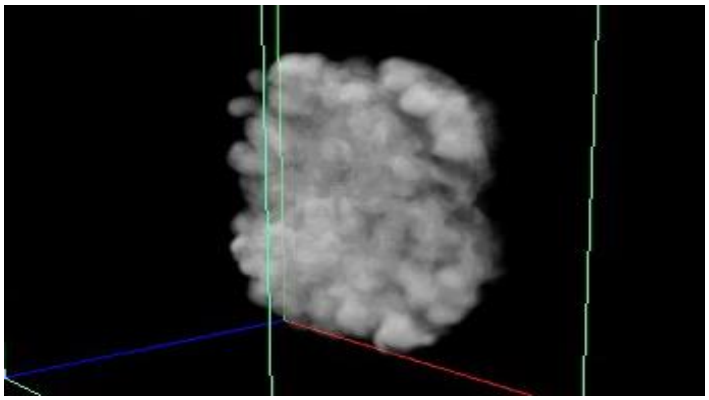


Figure 4.21 Simulation d'une poussière de verre à l'aide de fluides émis par des particules.

génération (*Inherit Velocity*). De cette façon, la poussière est émise à une grande vitesse et, afin qu'elle ralentisse assez rapidement, j'augmente la valeur de l'attribut lié à la perte de vitesse (*Damp*). La simulation nécessitera encore quelques ajustements lorsqu'il s'agira de le faire correspondre précisément aux autres effets, néanmoins nous possédons déjà à ce stade une bonne base de travail (***Figure 4.21***).

k. De nouvelles erreurs commises

En poursuivant mes tests de simulations pour ma poussière de verre, je me rends compte subitement de deux choses.

Premièrement, il va à priori être nécessaire de créer autant de simulations de Fluide et de *Setup* différents qu'il y a de valeurs et d'angles de caméra. En effet, il serait impossible en termes de puissance de calcul de créer une simulation qui soit suffisamment détaillée pour un plan serré, et qui couvre assez d'espace pour être visible en entier dans un plan large.

Deuxièmement, sans doute la plus grosse erreur que j'ai commise en voulant aller trop vite, je me rends compte qu'il faut rapidement régler la notion d'échelle — en réalité cela aurait dû être la première chose à faire. En effet, depuis le début de ma mise en place des débris, je me suis basé sur l'onde de choc réalisée au préalable, qui n'était elle-même basée sur rien. Ainsi, je me rends compte que j'ai omis un point assez indispensable, la mise en place de l'élément central autour duquel vont s'articuler tous les éléments de ma séquence, l'hélicoptère.

3.3.3. La déformation de l'hélice

a. Animation de l'hélicoptère et ajustement de l'onde de choc

Après avoir trouvé un modèle d'hélicoptère avec suffisamment de détails sur internet, je me lance dans la mise en place. La première étape est de régler la taille. Afin que mon travail jusqu'à présent n'ait pas servi à rien, je décide de mettre mon hélicoptère à l'échelle en me basant sur la taille de l'onde de choc réalisée au tout début. Pour l'animation, je sépare l'hélicoptère en trois éléments ; le corps, l'hélice principale et l'hélice arrière. J'anime à la main l'hélice principale et le déplacement de l'hélicoptère en m'inspirant évidemment de la scène de référence. En ce qui concerne l'hélice arrière, étant donné que je souhaite qu'elle tourne constamment, je crée une expression sur son paramètre de rotation: "*rotateX = time*500;*". Une fois l'hélicoptère mis en place, je me rends compte que l'onde de choc nécessite quelques ajustements. L'effet n'étant pas assez marqué au moment de l'impact, il est la taille de la vague doit être réglée pour qu'elle émerge plus rapidement. A présent, afin de finaliser l'animation, je dois prendre en considération l'encastrement et la déformation de l'hélice principale dans le bâtiment, et ajuster le mouvement de l'hélicoptère en fonction. Dans un souci de réalisme, je décide de réaliser cet effet de manière dynamique, à l'aide du plug-in DMM.

Figure 4.22 Modèle d'hélicoptère récupéré sur une banque de donnée 3D internet.

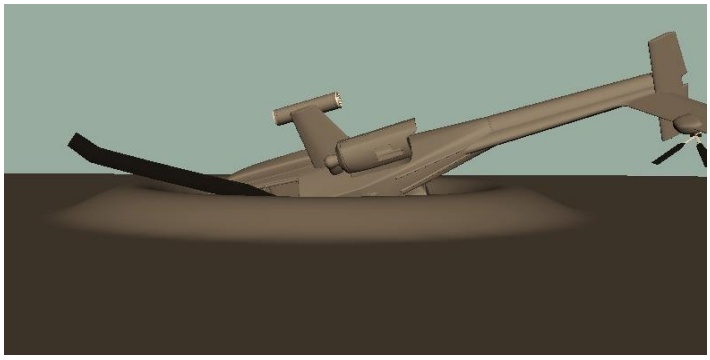


Figure 4.23 Onde de choc réajustée en fonction de l'animation de l'hélicoptère.

b. Création et paramétrage d'un objet DMM pour déformer l'hélice

Je convertis alors en objet DMM l'hélice qui rentre en contact avec l'immeuble. J'anime ensuite les paramètres pour que l'objet passe de passif à actif juste avant d'atteindre la surface. De cette façon le système n'a pas à calculer l'objet avant sa collision, ce qui accélère la vitesse de lecture. Afin de contraindre l'hélice au retors, je crée une *Glue Region* entre ces deux objets (***Figure 4.24***). Puis, pour que l'hélice puisse entrer en collision avec le bâtiment, je crée un cube DMM passif que je place juste derrière ma surface. Je passe alors aux réglages du matériel DMM de l'hélice. Pour que l'objet soit suffisamment inflexible, j'augmente la valeur de *Youngs* — rigidité de l'objet. Je le rends indestructible — paramètre *Toughness* à 0 — mais déformable — j'augmente la valeur de *Max Yield*. Enfin, concernant sa déformation, je veux qu'elle soit rapide — attribut *Creep* assez élevé — et définitive, c'est à dire qu'il ne retrouve pas sa forme initiale — paramètre *Yield* proche de 0.

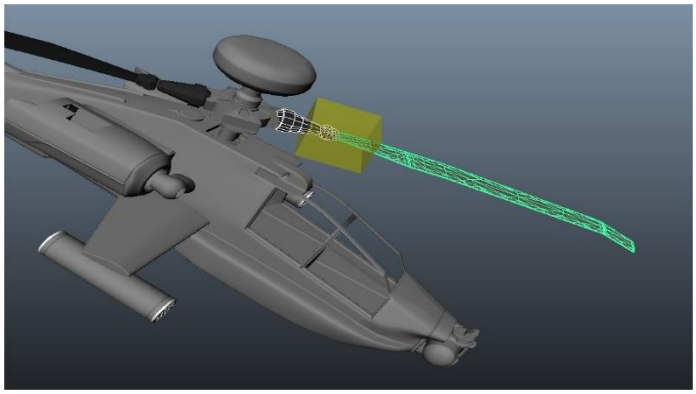


Figure 4.24 Hélice DMM contrainte au retors de l'hélicoptère à l'aide d'une **Glue Region**.

c. Réglage de la déformation de l'hélice avec l'immeuble

A la lecture, lorsque l'hélice entre en collision avec la surface, elle se déforme comme une feuille de papier et glisse le long de la surface. Je décide alors de supprimer certains *Edges* de ma géométrie afin qu'elle se déforme à moins d'endroits. Pour éviter que l'hélice ne glisse sur la surface, je tente d'augmenter les paramètres de friction des deux objets mais cela n'a que peu d'effet. Je crée alors un nouveau cube DMM passif que je place sous le point de collision pour bloquer l'hélice lors de sa déformation. Le résultat est déjà plus intéressant mais un nouveau problème apparaît. En effet, à

présent, lorsque l'hélice atteint la surface, elle pénètre l'immeuble, heurte la surface de collision, se tord puis se replie sur elle-même avant de réapparaître un peu plus bas (*Figure 4.25*). J'essaie alors d'animer mes cubes de collision pour qu'ils se retirent avant que l'hélice ne rebique, mais en vain. Je prends alors une solution plus simple, animer les cubes DMM passif pour qu'ils viennent déformer l'hélice comme je l'entends. Je parente alors les cubes au retors, les place de part et d'autre de l'hélice et les anime pour qu'ils la tordent au moment supposé de la collision. Je suis enfin satisfait du résultat.

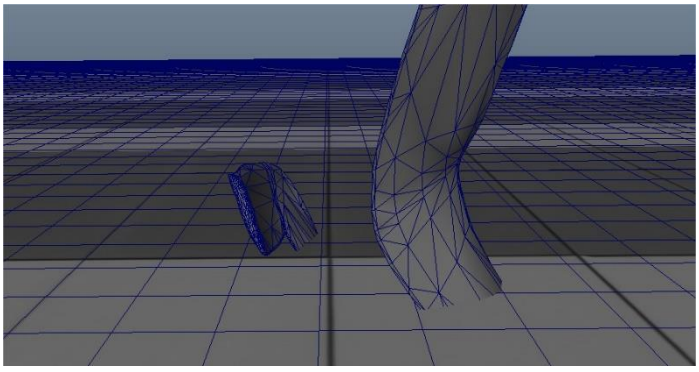


Figure 4.25 Hélice DMM qui traverse une surface avant de ressurgir plus bas.

3.3.4. Les éclats de vitres — Partie 2

a. Mise en place de l'hélicoptère et analyse des problèmes

J'importe alors l'animation de l'hélicoptère dans ma scène dédiée aux débris de verre. Je règle le cadrage de la caméra en m'inspirant des plans de Matrix. Il devient alors clair que ma simulation n'est pas du tout adaptée. En effet, la densité des particules — donc leur nombre — n'est pas suffisante et la texture d'émission n'est pas assez précise pour simuler tous les détails visibles dans la référence (*Figure 4.26*). De décide alors de chercher

Figure 4.26 Explosion des vitres, *The Matrix, Lana Wachowski, 1999*.

Figure 4.27 Détail de l'explosion des vitres, *The Matrix, Lana Wachowski, 1999*.

une nouvelle méthode d'émission. Afin de repartir sur de bonnes bases, je repense au problème depuis le début. Je souhaite créer des émissions successives de particules depuis des anneaux de plus en plus grands. En me penchant de plus près sur la séquence de référence, je me rends compte qu'à chaque explosion, les débris sont éjectés depuis trois anneaux (*Figure 4.27*). Il faut donc créer un système d'émission assez précis pour pouvoir reproduire ce phénomène.

b. Nouvelle solution pour l'animation des anneaux d'émission

Concernant l'animation, le fait de repartir sur de nouvelles bases me permet de prendre du recul, je réalise alors une nouvelle erreur que j'ai commise — une fois n'est pas coutume. En effet, je souhaitais dissocier le rayon des anneaux d'émission avec le timing — vitesse à laquelle on passe d'un anneau à l'autre pour l'émission. Or il est tout à fait possible de le faire avec de banales clés d'animation sur un simple torus. En effet, si le nombre de particules émises par secondes est constant, nous pouvons animer le rayon du torus à l'aide d'images clés par étape — ou *Stepped Key*. Le torus passera alors instantanément d'une taille à l'autre. De cette façon, si l'on souhaite modifier la vitesse à laquelle se succèdent tous les anneaux, il suffit de modifier la taille horizontale de la courbe et si l'on veut modifier le rayon des anneaux, il suffit de modifier la hauteur des points de la courbe d'animation (*Figure 4.28*).

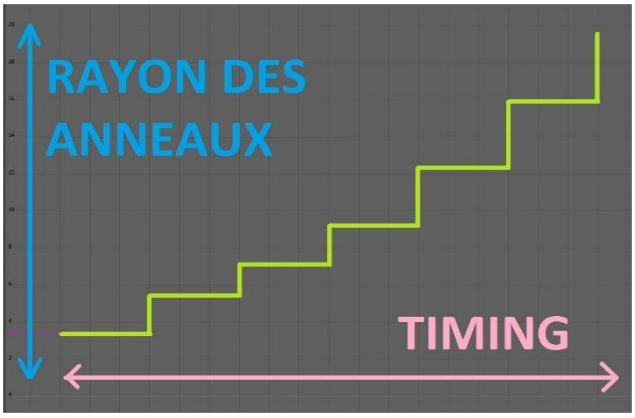


Figure 4.28 Courbe d'animation de la taille de l'anneau d'émission.

c. Création et modification d'un torus pour l'émission des débris

Pour l'émission des particules, je pense également à une méthode qui peut s'avérer plus simple. Je crée un torus avec suffisamment de subdivisions. Puis je supprime les faces qui ne m'intéressent pas de telle sorte qu'il ne me reste plus que trois bandes en forme d'anneaux (*Figure 4.29*). En gardant l'historique de construction, j'ai accès à deux paramètres toujours actifs, le rayon du torus et son épaisseur, parfait pour paramétrer et animer notre effet. L'avantage de cette méthode par rapport à l'extrude d'une ligne le long d'un cercle, est que je vais avoir des *UV* fixes, puisque je n'anime pas de paramètres de création, ce qui sera intéressant lorsque j'ajouterai une texture d'émission.

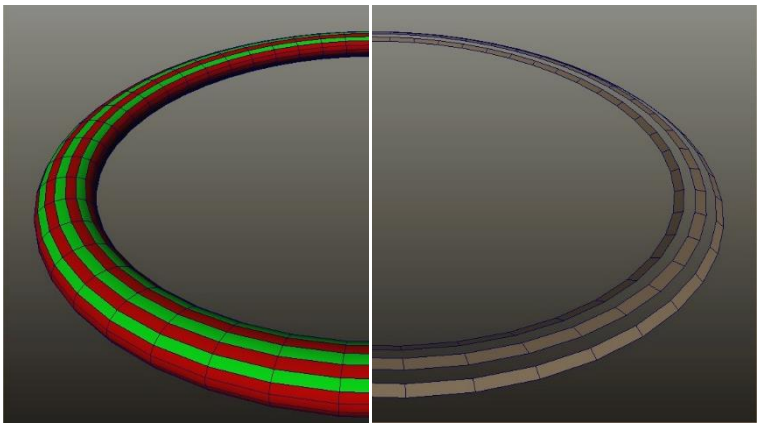


Figure 4.29 Modification d'un torus. Les faces vertes sont conservées, les rouges supprimées pour créer trois anneaux. La partie droite représente le torus modifié.

d. Emission et perturbation des particules depuis le torus

Je crée alors un système de particules que j'émets depuis cette géométrie. Afin de donner une impulsion initiale, j'augmente la vitesse le long de la normale. A la lecture, la simulation est trop propre, les anneaux d'émissions sont trop marqués. J'applique alors à l'émetteur un *Noise* pour la texture d'émission. Le résultat est en effet moins uniforme mais les cercles sont encore trop visibles. Après différents essais, je réussi finalement à casser davantage cette forme, en venant perturber le torus à l'aide de deux *Wave Deformers*. Le problème est qu'à présent, chaque anneau d'émission se ressemble fortement, même après avoir animé le *Noise*. Je décide alors d'animer les

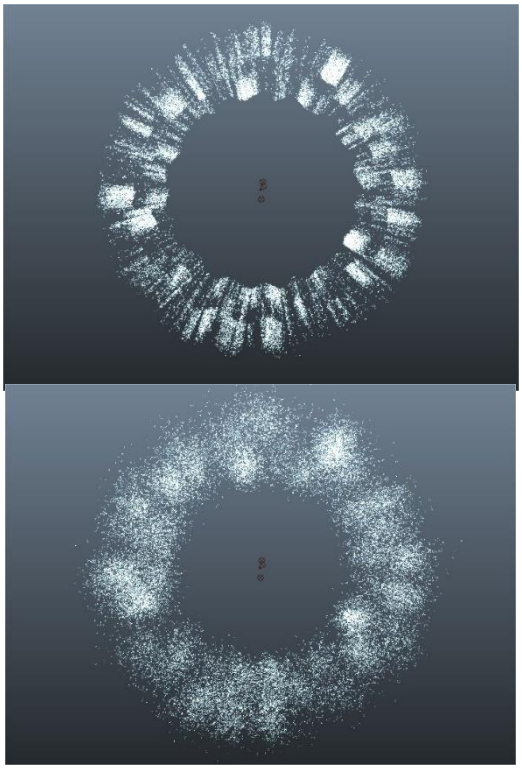


Figure 4.30 En haut, des particules émises depuis un torus. En bas, les mêmes particules, perturbées par une turbulence avant leur apparition.

déformeurs en rotation pour qu'ils soient toujours différents. Enfin, pour casser définitivement leur forme, j'utilise une petite astuce. Je déplace le torus d'émission à quelques unités derrière la façade de l'immeuble, puis je crée une turbulence sous forme de volume que je positionne juste entre ces deux objets. De cette façon, lorsque les particules sont émises, elles passent par un champ de force qui les perturbe, puis apparaissent à la surface du bâtiment avec un aspect moins uniforme (*Figure 4.30*). Cependant, notons qu'au rendu des particules, la façade de l'immeuble sera masquée. Par conséquent, nous verront les particules être émises depuis le torus puis être perturbées. Ainsi, afin d'anticiper ce problème, je décide de créer une expression sur mes particules permettant de ne les afficher qu'une fois qu'elles sont ressorti du champ de force (*Figure 4.31*). L'émission étant enfin prête, je réintègre les systèmes de particules, de fluides et les champs de force déjà paramétrés en les liant à ce nouvel émetteur.

Expression:

```
vector $pos = position;
if($pos.z<0)
opacityPP = 0;
else
opacityPP = rand(0.001,0.3);
```

Figure 4.31 Expression qui change l'opacité de particules en fonction de leur position.

3.3.5. L'explosion de l'hélicoptère

L'étape la plus délicate est à présent de mettre en place l'explosion de l'hélicoptère. Étant donné que les fluides peuvent rapidement s'avérer gourmand en termes de ressources, il est important qu'il y ait le moins d'éléments possible dans ma scène. J'en crée alors une nouvelle sans oublier néanmoins d'importer mon animation d'hélicoptère — il s'agit de ne pas reproduire les mêmes erreurs. Je crée donc un conteneur de fluide d'une taille correcte pour commencer mes tests. J'active directement le calcul dynamique de la température et du combustible (**Fuel**), composantes pratiquement indispensables lors de la simulation de ce genre d'effets.

e. Création de particules pour émettre le fluide

J'ai pu remarquer au fil de mes différents tests, que l'émission de fluides depuis des systèmes de particules pouvait donner lieu à des résultats plutôt satisfaisant en termes de forme et de dynamique. Mon premier réflexe est donc de me lancer dans cette direction — d'autant plus que dans notre référence, l'explosion dégage des boules de feu qui sont envoyés de toute part, ce qui correspondrait à priori à ce genre de dynamique. Je crée alors une émission de particules en deux parties. Un premier système qui envoie une poignée de particules. Et un deuxième émis depuis les premières va de telle sorte à obtenir plusieurs épaisses trainées bien distinctes (*Figure 4.32*). J'utilise ensuite les secondes particules comme émetteur des trois composantes de mon fluide — densité, chaleur et carburant.

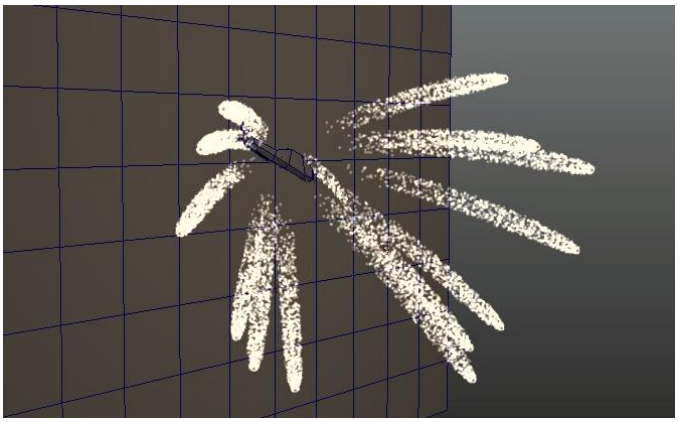


Figure 4.32 Système de particules qui va servir d'émetteur au futur fluide.

f. Réglages du fluide et constatation de problèmes

Le résultat est assez intéressant visuellement au moment de l'explosion — quoiqu'un peu trop propre au niveau de la forme —, mais devient cependant assez étrange dès que toute la chaleur s'est dissipée (*Figure 4.33*). Afin de créer une simulation un peu plus désordonnée, avec des trainées moins marquées, je connecte à mes particules et mon fluide une turbulence sous forme de volume que je dispose près de la surface de l'immeuble. Je réalise également quelques ajustements au niveau du paramètre de *Swirl* de mon fluide et en ce qui concerne la direction, la vitesse, et la quantité de fluide émise. Malgré tout, comme dans ma première simulation (*Figure 4.33*), un problème persiste. Une fois que la chaleur s'est dissipée, les différentes trainées générées se retrouvent transformée en fumée noire plus ou moins stagnantes, produisant un résultat plutôt disgracieux. Je tente alors d'augmenter la *Buoyancy* du fluide pour qu'il s'élève plus rapidement une fois refroidi. Le résultat est encore pire. Étant donné que ce sont les particules qui conduisent la chaleur, l'explosion est contrainte le long des différentes trainées au moment de la détonation, puis, dès que la chaleur disparait — en même temps que les particules — le fluide se met brutalement à s'élever dans les airs, cassant complètement le rythme et la direction initialement instaurée. Je tente alors d'appliquer aux particules une gravité inverse afin qu'elles s'élèvent au moment

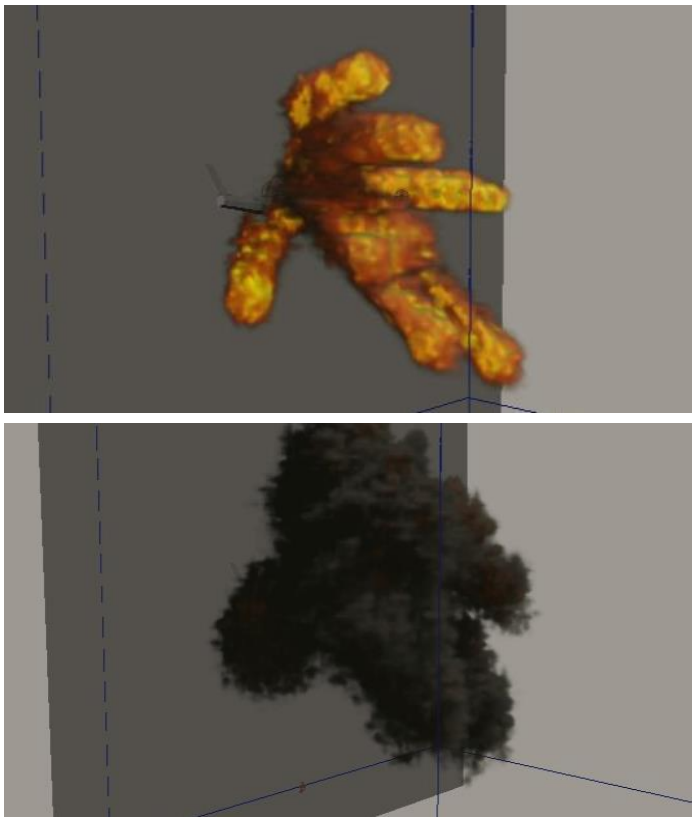


Figure 4.33 Explosion générée par un fluide émis à partir de particules. En haut, la simulation après 3 secondes d'émission, en bas après 6 secondes.

de l'explosion, dirigeant ainsi les boules de feu vers le haut du conteneur. Je ne réussis malheureusement pas à obtenir un résultat satisfaisant, même en ajoutant un champ de force pour pousser globalement le fluide vers le haut. Comme je me rends compte que je passe beaucoup trop de temps à tenter de régler cet effet sans résultats convenables, je décide d'expérimenter une nouvelle méthode d'émission. Pour repartir sur des bases saines, je souhaite me concentrer à présent sur un émetteur de fluide classique que j'essaierai de paramétrer pour obtenir le résultat escompté.

g. Création d'un émetteur de fluide classique

Je crée alors un émetteur de type volume sphère que je positionne au niveau de l'hélicoptère. Afin de donner cette sensation de puissance et de vitesse lors d'une explosion, plusieurs paramètres vont rentrer en compte. En premier lieu, l'émetteur devra être configuré pour générer une importante quantité de chaleur et de carburant pendant une assez courte durée. La densité peut également être paramétrée pour n'être émise qu'au moment de l'explosion, ce qui résultera en un simple nuage de fumée qui va s'évaporer puis finalement disparaitre. Dans mon cas, après l'explosion, je souhaiterais que l'hélicoptère continue d'émettre de la fumée. Je maintiens donc l'émission de la densité.

h. Réglages de l'émission de l'explosion

Afin de créer le souffle qui va violemment pousser l'explosion au moment de la détonation, il existe plusieurs techniques. La première consiste à venir modifier les paramètres de vitesse au niveau de l'émetteur du fluide. Je règle alors la vitesse directionnelle de mon émetteur pour qu'il envoie la simulation dans la direction opposée à la surface de l'immeuble (*Figure 4.34*). Étant donné qu'après l'explosion je continue à émettre de la densité, j'anime mon paramètre de vitesse pour qu'il pousse mon fluide uniquement au début de la simulation. Le résultat est cependant peu concluant car mon fluide se retrouve éjecté dans une seule direction ce qui crée un effet trop rectiligne et uniforme (*Figure 4.35*). Je tente alors d'utiliser un champ de force de type Radial ayant justement la particularité d'émettre une force dans toutes les directions à la manière d'un émetteur de particules omnidirectionnel. Le résultat est un peu plus intéressant car le fluide se retrouve propulsé de façon moins linéaire, cependant sa forme est toujours trop propre, trop parfaite. Afin de perturber cette forme, j'active tout d'abord la turbulence d'émission dans les paramètres de mon émetteur afin que la densité du fluide émise ne soit pas la même sur toute la surface du volume d'émission. Le résultat n'étant pas suffisamment irrégulier, je devrais utiliser en plus un champ de force turbulence qui

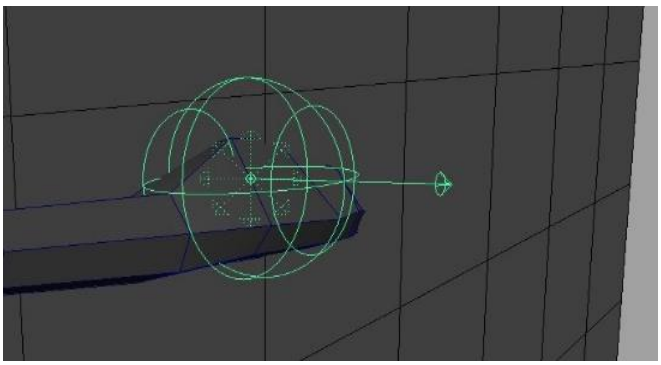


Figure 4.34 Emetteur de fluide de type volume avec une vitesse directionnelle.

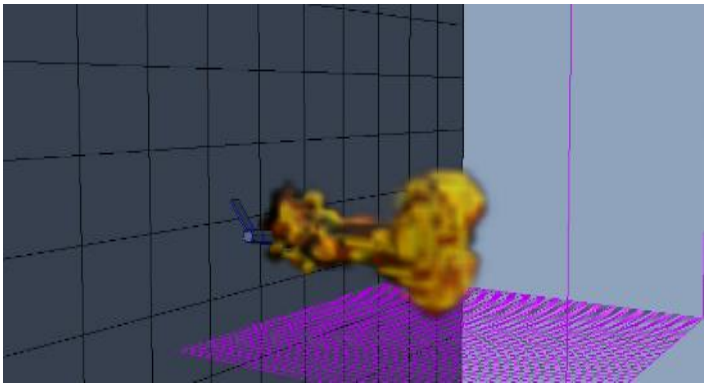


Figure 4.35 Fluide émis depuis un émetteur avec une forte vitesse directionnelle.

viendrait affecter le fluide au moment de son émission. Cependant, pour des questions de vitesse de calculs, il est préférable autant que faire se peut d'utiliser le moins de champs de force possible — et je sais qu'il me faudra en rajouter d'avantage tôt ou tard. Je remplace alors mon *Radial Field* par un champ de force de type *Volume Axis*, la star des champs de force, qui possède une pléthore de paramètres empruntés à tous les autres — l'attribut turbulence en fait partie. J'anime alors le paramètre *Away From Axis* — qui confère un mouvement de type radial — et l'attribut Turbulence, en leur donnant une forte valeur au moment de l'explosion. J'obtiens enfin un résultat plus intéressant (*Figure 4.36*).

Figure 4.36 Fluide émis avec une forte vitesse directionnelle et de la turbulence.

e. Réglages du timing de l'explosion

À présent que le fluide possède une bonne émission et forme initiale, il s'agit de le faire évoluer correctement dans le conteneur de la façon la plus analogue possible à notre référence. Première constatation, le timing n'est pas bon. En effet, bien que la forme initiale soit correcte, le Fluide n'est pas émis suffisamment rapidement et ne ralentit pas non plus à mesure qu'il s'élève dans les airs. C'est donc la première chose à régler. Pour ce faire, je joue avec deux paramètres liés à la vitesse du Fluide. En augmentant, tout d'abord, l'attribut *Damp*, je fais perdre au Fluide de la vitesse à mesure que la simulation s'écoule. Puis, en animant le paramètre *Simulation Scale Rate*, qui gère la vitesse de simulation, je peux donner un petit coup de pouce au fluide pour qu'il soit émis plus rapidement et ralentisse progressivement.

f. Paramétrage de la taille du fluide

Le temps est maintenant venu de commencer à régler la résolution et l'échelle du Fluide. En effet, actuellement, l'explosion semble un peu trop grande par rapport à l'hélicoptère et s'élève trop rapidement et trop haut dans les airs. Plusieurs attributs vont me permettre de régler ce problème et, en même temps, d'accroitre la qualité de mon fluide. Le premier, bien entendu, est la résolution. En augmentant la résolution du fluide, j'augmente son nombre de *Voxels*, donc j'affecte sa précision et la façon dont il se déplace. C'est donc ce que je fais (*Figure 4.37*) ; sa qualité visuelle se trouve donc accrue en même temps que son échelle globale est réduite. Autre paramètre qui modifie également considérablement la simulation et qui me sera bénéfique, l'attribut *Substeps* — attribut permettant de régler la précision des calculs de simulation. En augmentant sa valeur, le fluide est plus précis, les volutes sont plus fines, mieux calculées et la simulation dans son ensemble semblera plus grande — car sera plus petite à l'écran — et s'élèvera moins haut. Néanmoins, je fais attention à ne pas trop l'augmenter, auquel cas le fluide perdrait trop de dynamique et ne monterait plus assez haut.

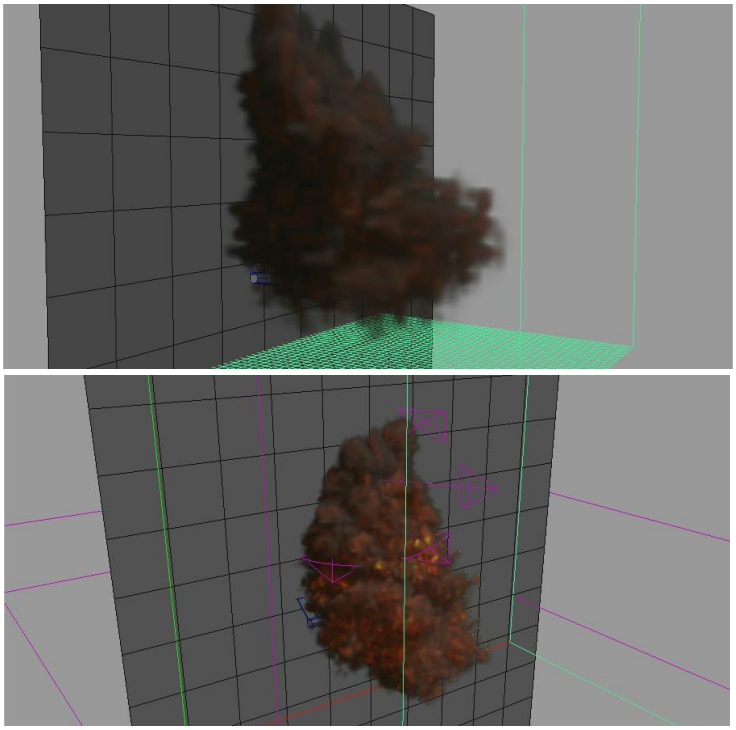


Figure 4.37 Le fluide du bas a été émis avec une résolution plus importante que celui du haut.

g. Contraindre le fluide dans l'espace

Malgré tout, mon fluide s'élève toujours trop haut et, forcément, finit par sortir du conteneur. Cependant, je voudrais essayer de le contenir à l'intérieur. Pour cela je crée une série de champs de force dans le but de le diriger. Dans la partie supérieure du conteneur, je crée un champ de force *Drag*, qui permet de ralentir le fluide lorsqu'il rentre dans son champ. La création d'un tel champ de force permet de ne pas avoir à ajouter du *Damp* sur ma simulation, ce qui lui ferait perdre de la puissance lors de son émission. Je constate également que le Fluide s'échappe par moments sur les côtés de mon conteneur. Comme je ne souhaite pas créer un fluide plus grand en élargissant sa zone d'émission, je crée deux *Uniform Fields*, des deux côtés du conteneur qui poussent chacun dans la direction opposée au bord. Ainsi, lorsque la simulation s'approche d'un peu trop près d'un rebord, elle est freinée par une force inverse, lui empêchant de sortir du conteneur (***Figure 4.38***).

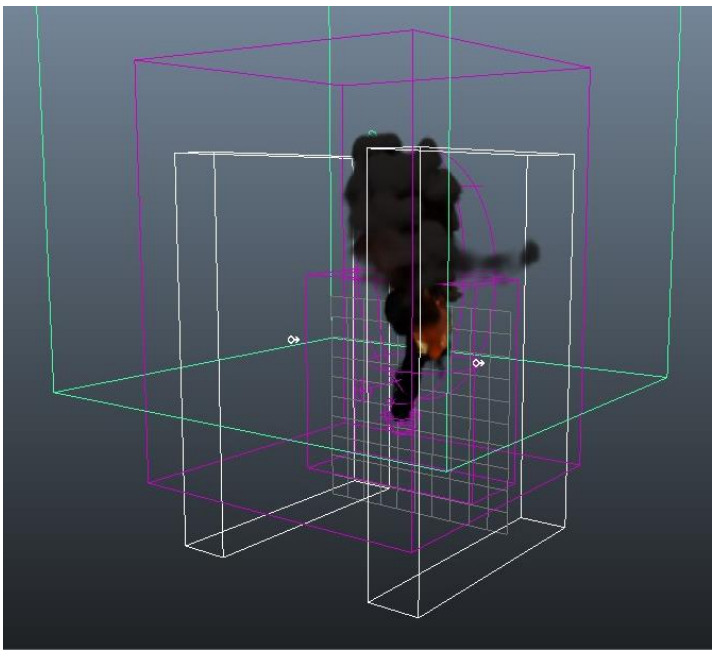


Figure 4.38 Fluide contrôlé par une série de champs de force. En vert un Drag Field, et en blanc deux Uniform Fields.

À suivre…

3.4. Compte-rendu

À l'heure actuelle, je pense pouvoir dire qu'il reste sans doute encore la moitié du travail à réaliser si je veux pouvoir arriver au bout de ce projet, qu'il s'agisse d'ajustements, de parties non-terminés ou d'effets que je n'ai tout simplement pas eu le temps de commencer. Je pense notamment à la légère fumée émanant de l'hélicoptère avant sa collision ou à la destruction de la façade de l'immeuble que j'avais l'intention de simuler à l'aide de DMM. En ce qui concerne les rendus, quelques tests ont déjà été effectués mais rien n'est pour le moment définitif (*Figure 4.39*). Il est vrai qu'il m'est arrivé de perdre du temps sur certaines parties soit car je n'utilisais pas la bonne méthode, soit parce que je voulais aller trop vite et sautais des étapes importantes. Écrire ce mémoire fut aussi l'occasion de prendre du recul sur mon travail et m'a parfois fait mettre le doigt sur des problèmes ou des évidences que je n'avais pas vu jusqu'à lors. Quoi qu'il en soit, la suite s'annonce explosive !

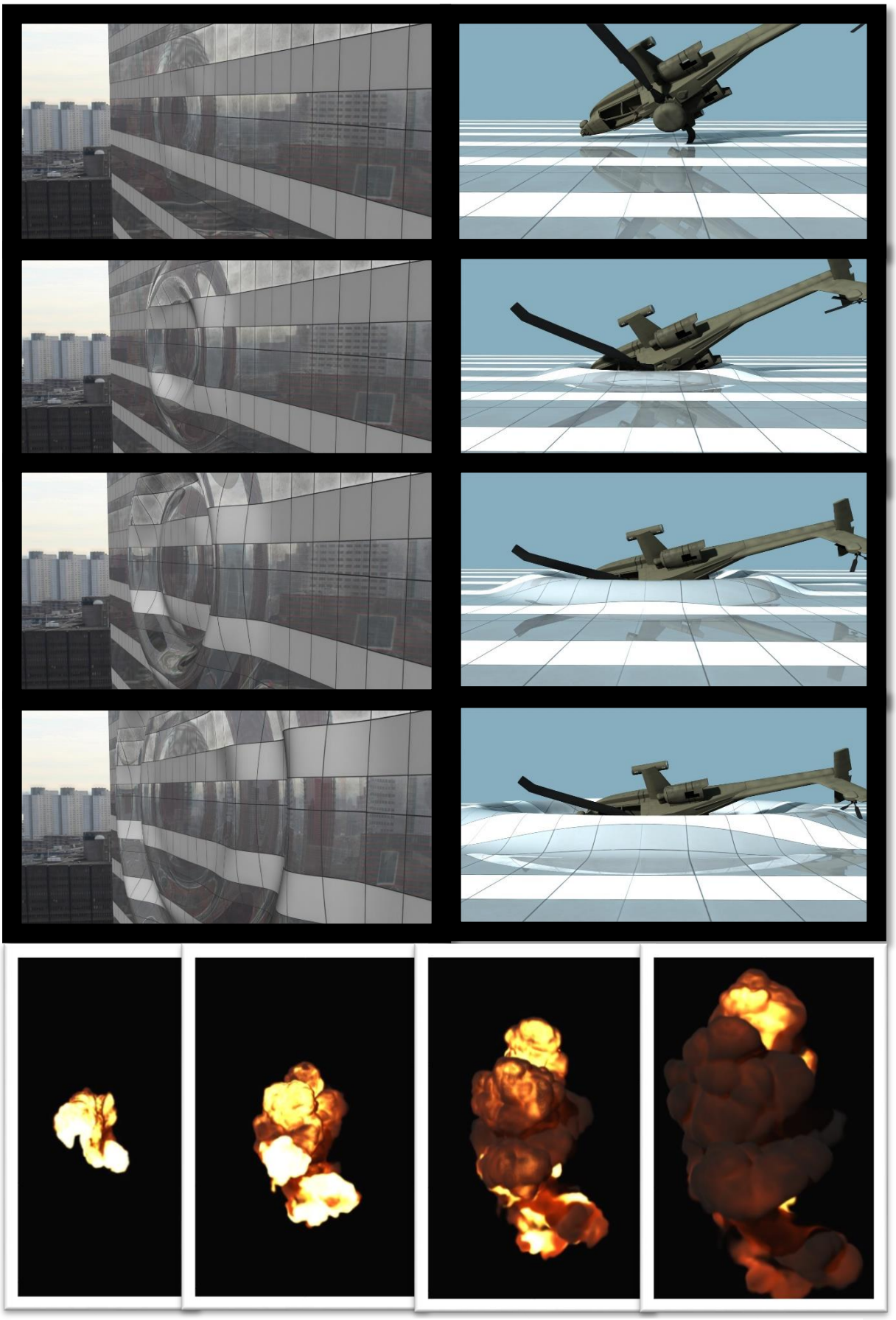


Figure 4.39 Tests de rendus

Conclusion

Comme je m'en doutais, réaliser un effet pyrotechnique convainquant n'est pas une tâche facile. Elle requiert à la fois des connaissances techniques assez pointues, mais surtout, un savoir-faire et des réflexes qui viennent avec le temps, à force de travailler avec les outils. Avant cette année, mes connaissances et expériences dans ce domaine étaient quasi-nulles. Bien qu'en un an je ne sois pas devenu un expert en la matière, je peux néanmoins constater une évolution lorsque je compare mes premiers tests aux plus récents. Cependant, au vue de l'avancement actuel de mes recherches exposées dans ce mémoire, je souhaiterais exprimer quelques déceptions. Tout d'abord, je pense ne pas avoir réussi à cibler mes objectifs à temps, ce qui m'a fait perdre un temps considérable. Je savais dans quelle direction allait se tourner ma problématique, néanmoins je ne savais pas comment l'aborder. Avant de décider de me restreindre à *Maya*, j'ai commencé à effectuer quelques tests sur *Houdini* et *3dsMax*. Comme il s'agit de logiciels que je n'utilise jamais, le langage de base n'était pas acquis et il aurait fallu, pour bien faire, s'investir complètement dans l'un deux afin de reprendre les bases nécessaires pour aller ensuite plus loin. Autre déception, je me rends compte avec du recul que je n'ai pas effectué autant de tests de destructions DMM que je l'aurais souhaité. J'ai bien réalisé quelques essais mais il est vrai qu'avec un sujet aussi vaste je n'ai finalement pas testé le système autant qu'il était prévu.

Pour finir, j'effectuerai un rapide parallèle entre Fluides pré-calculés et effets temps réel. En effet, après avoir vu récemment la bande annonce pour le future jeu vidéo *Battlefield 4*, j'ai été, n'ayons pas peur de le dire, émerveillé par la qualité des effets spéciaux, notamment au niveau de la dynamique des explosions. En temps normal, lorsque l'on souhaite créer une explosion dans un jeu, il est fréquent d'utiliser un système de particules qui s'affichent en *Sprites* — chaque particule est remplacée par une image en deux dimensions —, et le résultat est généralement loin de ressembler à une simulation de Fluides pré-calculée. Cependant, dans cette bande annonce, les

explosions sont absolument renversantes car elles ressemblent à des simulations de Fluides, sauf qu'elles sont calculées en temps réel ! Bien qu'il n'existe aucune information disponible sur ce jeu pour le moment, en effectuant quelques rapides recherches je suis tombé sur ce qui pourrait être un début de piste. Lors d'un projet collaboratif entre *Nvidia*, *Passion Pictures* et Le moteur de jeu *Unity* — *The Butterfly Effect* —, des artistes ont créés une explosion à l'aide de surfaces dynamiques plutôt que de particules ce qui donne un résultat qui ressemble beaucoup plus à un Fluide qu'à ce que l'on a l'habitude de voir dans d'autres simulations en temps réel. Il me prend alors à penser que si ce système se perfectionne, pourquoi ne pas imaginer pouvoir l'utiliser pour des séquences pré-calculées, et donc gagner un temps précieux de calcul. Néanmoins, rien ne nous dit qu'il s'agisse de cette méthode qui est utilisée dans *Battlefield 4*. Espérons que nous en sauront rapidement d'avantage.

Battlefield 4, développé par Dice, édité par Electronic Arts, 2013.

Bibliographie

Richard RICKITT, ***Special Effects, the history and technique***, Billboard Books, 2007, 312p.

Todd PALAMAR, ***Maya Studio Projects Dynamics***, Wiley Publishing, Inc, 2009, 275p.

Alan MCKENZIE, ***Trucages et effets spéciaux au cinéma***, Paris, Editions Atlas, 1987, 167 p.

Roger HICKS, ***Effets spéciaux***, Paris, Editions VM, cop. 1996, 159 p.

Ron GANBAR, ***Nuke Professional Compositing and Visual Effects***, Peachpit Press, 2008, 386p.

Ron BRINKMANN, ***The Art and Science of Digital Compositing***, Elsevier Libri, 2008, 704p.

Patricia D. NETZLEY, ***Encyclopedia of Movie Special Effects***, Greenwood Press, 1999, 306p.

John A. CONKLING, Christopher J. MOCELLA, ***Chemistry of Pyrotechnics Basic Principles and Theory Second Edition***, CRC Press, 2011, 226p.

Webographie

http://www.thegnomonworkshop.com/store/product/898/Introduction-to-Maya-Fluid-Effects-.Uah2U0D0F8E

https://vimeo.com/vfxlearning

http://www.pyragricnordest.fr/cms/index.php?option=com_content&view=article&id=1&Itemid=2

http://www.fracture-fx.com/

http://www.pandora-pyroshop.com/blog/histoire-de-la-pyrotechnie/

http://lesterbanks.com/maya-tutorials/maya-dynamics-tutorials/page/2/

http://download.autodesk.com/global/docs/maya2013/en_us/index.html