Stranger
Fansub

Zeste de savoir

La 3D pour le jeu vidéo avec Blender

12 août 2019

Table des matières

Vous avez un projet de jeu vidéo, mais vous débutez en 3D ? Vous êtes un habitué de la modélisation mais les textures vous font peur ? Vous êtes simplement curieux de découvrir les méthodes des pros ?

Ce cours est fait pour vous ! A travers un objet d'exemple, nous allons voir pas à pas comment un élément de jeu vidéo est créé aujourd'hui. Le cours s'appuie sur le modeleur libre Blender et nécessitera également l'usage d'un logiciel de retouche d'image tel que Adobe Photoshop ou GIMP .

FIGURE 0. – L'exemple de ce cours ; les talents artistiques de l'auteur seront excusés

1. A propos du cours

Ce cours se veut **simple et accessible**. L'objectif est de vous apprendre une méthode typique pour réaliser un objet de jeu vidéo, à travers un exemple.

Voici **ce que ce cours n'est pas** :

— un cours de modélisation ;
— un guide pas-à-pas qui vous rappelle le détail des opérations ;
— un cours universel pour créer tous les objets possibles ;
— une encyclopédie qui explique tout.

On va voir ici des techniques très modernes puisque le cours se base sur la technologie du **Physically Based Rendering**. Un moteur de jeu récent comme Unreal Engine 4, CryEngine 3 ou Unity 5 est indispensable ici. Le monde du logiciel libre ne propose pas encore ce stade d'évolution technique.

Une connaissance basique de Blender est requise : modélisation simple, *modifiers* et *UV mapping*. De nombreux cours de qualité existent sur ce sujet, à commencer par celui-ci .

2. Introduction : le Physically Based Rendering

Pour commencer ce cours, faisons le point sur la technologie.

Votre ordinateur est équipé d'un processeur sur lequel s'exécutent des programmes, mais il dispose également d'un processeur graphique, le GPU, sur votre carte graphique. Ce GPU accueille également des programmes : on les appelle les **shaders**, et ils produisent les pixels à l'écran à partir d'objets 3D, de textures et de paramètres divers.

Les shaders sont difficiles à réaliser et doivent être écrits par un programmeur spécialisé, c'est une tâche assez pénible qu'on cherche à éviter. Aujourd'hui, on préfère donc laisser des programmes générer ces shaders automatiquement à partir d'une définition, le **matériau**. Un matériau est un ensemble de paramètres qui peuvent être manipulés directement par un artiste, sans écrire de code, pour contrôler l'aspect d'une surface.

Figure 2. – Un matériau dans Unreal Engine 4

Dans les années 2000, les jeux vidéo ont introduit de nombreuses techniques pour réaliser les plus beaux rendus possibles, tout en se pliant aux contraintes du matériel, et jusqu'à 2010, il était difficile de réaliser un objet dont le rendu soit crédible par rapport à une photo. On travaillait alors avec des modèles très approximatifs de la matière (pour les curieux, on parlait alors de **diffuse** et **specular** pour désigner la couleur et les reflets d'un objet).

Le **Physically Based Rendering** ou **PBR** est un modèle de **matériau** pour créer des rendus 3D qui respectent les lois de la physique. Introduit d'abord dans le monde des films d'animation, ce modèle s'est introduit en quelques années dans les principaux moteurs de jeu moderne.

Dans un matériau en PBR, on a généralement trois paramètres centraux :

— l'**albedo** est tout simplement la couleur de la surface ;
— la **roughness** contrôle la rugosité de la surface, du miroir parfait au papier de verre ;
— la **metalness** définit si la surface est un métal.

FIGURE 2. – Différentes valeurs de roughness et metalness

Sur l'exemple ci-dessus, le matériau a les propriétés suivantes, de gauche à droite :

— metalness = 0, roughness = 0 ;
— metalness = 1, roughness = 1 ;
— metalness = 1, roughness = 0 ;
— metalness = 1, roughness = 0.5.

L'avènement des modèles physiques est une excellente nouvelle pour vous : créer des rendus réalistes n'a jamais été aussi facile.

3. Liste des courses

Avant de se lancer en cuisine, on va faire le point sur les ingrédients. Pour réaliser un objet dans un jeu vidéo, au niveau de qualité attendu aujourd'hui par un joueur, il va nous falloir :

— un objet en 3D finement détaillé dans votre modeleur favori, c'est ce qu'on appelle un **high poly** ;
— une version plus légère de cet objet, le **low poly** ;
— un matériau PBR dans votre moteur de jeu favori ;
— plusieurs **textures**.

? Pourquoi a-t-on besoin de plusieurs textures ?

Autrefois, on utilisait une seule texture, celle qui correspondrait aujourd'hui à l'**albedo**. Aujourd'hui, on va utiliser différentes textures pour contrôler le rendu :

— une **albedo map**, qui définit l'albedo, la couleur de l'objet ;
— une **normal map** qui simule des reliefs ;
— une **roughness map** ou **gloss map** en noir et blanc pour contrôler la roughness ;

— une **ambient occlusion map** qui assombrit les recoins de vos objets, également en noir et blanc.

La metalness, pour cet objet, sera contrôlée par une simple valeur fixe : tout l'objet est métallique, il est donc inutile d'utiliser une texture pour ça. Dans un matériau, on peut utiliser aussi bien une texture qu'une valeur uniforme sur toute la surface !

Pour terminer les présentations, reprenons l'exemple de notre valve...

FIGURE 3. – Dans l'ordre de lecture : albedo, normal, roughness et ambient occlusion

? Il faut donc réaliser ces quatre textures et les associer au matériau de l'objet ?

Tout à fait ! La façon exacte de faire dépend énormément du jeu et de la technologie qui va avec, mais les moteurs de jeu cités en avant-propos proposent des moyens simples de créer un matériau à partir des textures.

4. Le high poly

La particularité d'un **high poly**, la version *haute définition* de votre objet, c'est que cet objet ne sera jamais présent dans le jeu. Le nombre délirant de faces, la complexité des *modifiers*, la difficulté à texturer sont autant de raisons. On va donc tricher, importer le low poly dans le jeu, et utiliser une technique qui permet de reproduire une partie des détails du high poly sur le low poly : le **normal mapping**.

FIGURE 4. – Le high poly : 96 000 faces après les modifiers, qui dit mieux ?

Le principe du normal mapping est le suivant : une texture, la **normal map**, va stocker les variations de la surface entre le high et le low. Le matériau va ensuite reproduire ces différences de surface une fois dans le jeu, en trichant sur l'éclairage. Le résultat est saisissant et permet de représenter des détails sans utiliser de géométrie supplémentaire dans le low poly.

FIGURE 4. – De gauche à droite : high poly, low poly et rendu ; la normal map est visible en bas

Dans cet exercice, le lissage des faces est votre principale contrainte. Un objet réel n'a que rarement des arêtes vives, sauf si il s'agit d'un couteau ou d'une pièce métallique fraîchement usinée. Arrondir les angles est donc la première mission de votre high poly, et pas qu'un peu : si l'objet réel a un arrondi de 5 millimètres de large, il faut au moins un centimètre d'arrondi sur le high poly pour avoir le même résultat final. C'est une contrainte qui est liée à la normal map : souvent, la zone autour d'une arête tient dans quelques pixels de cette texture, un lissage très net sera tout simplement invisible.

Il faut donc modéliser en tenant compte de cette contrainte, en exagérant les formes que vous souhaitez introduire dans la normal map. Les rivets de l'exemple ont une forme sphérique, mais au rendu ils n'existent que par une distorsion de l'éclairage et ont donc l'air quasiment plats.

Il y a deux outils dans Blender pour contrôler le lissage d'un high poly :

— l'outil *edge loop* (CTRL + R) sert à diviser des faces, on peut s'en servir pour limiter la propagation du lissage, comme illustré ci-dessous : on appelle ça une *control loop* ;
— l'outil *edge crease* (dans le menu de la vue 3D, affiché avec N) s'applique à des edges et définit leur mode de lissage, de 0 à 1 : 0 pour ne pas influencer le lissage, 1 pour créer une arête vive.

FIGURE 4. – Les arêtes lissées : observez leurs edge crease (en violet) et les control loop (en vert)

Dans la mesure du possible, il faut soigner le lissage en connectant le maillage d'une partie à l'autre, sans discontinuité. Bien sûr, la patience a ses limites et vous pouvez aussi recourir à de la géométrie dite *flottante*, c'est-à-dire qui n'est pas reliée au reste de l'objet, si :

— soit la différence n'est pas visible et c'est donc inutile (les rivets ci-dessus) ;
— soit il ne faut pas de lissage car vous représentez des pièces séparées (entre l'écrou et la manette sur notre valve, par exemple).

?

Toutes ces heures de modélisation, c'est juste pour faire une seule texture ?

Presque : on génèrera en fait une deuxième texture à partir du high poly. Mais en effet, il ne va servir qu'à générer des ressources et ne sera jamais importé dans le jeu. Rassurez-vous, le temps

passé en modélisation n'est pas perdu : une belle normal map transforme totalement votre low poly.

5. Low poly et UV mapping

Le low poly est la version light de votre objet ! On va voir ensemble comment le réaliser de façon efficace. D'abords, parlons un peu de ses contraintes :

— il doit épouser parfaitement les courbes de votre high poly, de préférence en les enrobant ;
— il doit minimiser le nombre de faces ou *polycount* ;
— il ne doit pas utiliser de subsurface ou autre déformation automatique qui serait *appliquée* à l'exportation, multipliant le *polycount* ;
— bien entendu, il doit comporter un UV mapping.

FIGURE 5. – 3 000 triangles, pour cet objet, c'est un poids raisonnable

Chacun a sa propre méthode pour le faire, la mienne est très rapide, au détriment peut-être du polycount. Voici la marche à suivre :

— dupliquez le high poly, supprimez le subsurface et ajoutez un modifier **edge split** avec un angle bien choisi pour votre objet, typiquement 45° ;
— supprimez toutes les *control loop* qui servent à contrôler le lissage : le low poly ne doit avoir des arêtes que là où la géométrie change ;
— n'hésitez pas à supprimer des jointures entre différentes pièces pour utiliser une géométrie flottante à la place, comme illustré ci-dessous ;
— n'oubliez pas de créer des *seams* dans des endroits peu visibles pour préparer l'**UV mapping**.

FIGURE 5. – Exemple de passage à une géométrie flottante, sur un autre objet

Vous pouvez utiliser des modifiers pour accélérer le travail, notamment le *mirror*, mais vous devez prendre bien soin de les appliquer avant de passer à l'UV mapping, à l'exception de l'*edge split* qui va nous accompagner jusqu'à l'exportation.

L'**UV mapping** est une étape pénible mais nécessaire, qui doit être réalisée avec minutie : si il y a de larges zones vides sur cette UV map, c'est autant d'espace perdu sur vos textures.

Attention cependant à ne pas trop condenser non plus, en particulier, vous ne devez jamais avoir deux faces superposées, même si ce sont des pièces rigourement identiques : la génération des textures va créer des artefacts très visibles si vous vous y risquez.

Une fois l'UV mapping terminé, votre objet est prêt pour être exporté. Le format de référence est **FBX**, il est supporté dans un plugin de Blender fourni par défaut que vous devez activer dans les préférences de l'outil. Au moment de l'exportation, prenez garde à sélectionnez votre low poly et à cocher la case **selected only**, pour ne pas exporter également le high poly, au risque de bloquer Blender pendant plusieurs minutes !

6. En cuisine pour le baking

Le **baking** est l'étape centrale du travail de création. Ce processus permet, à partir du high poly et du low poly, de produire automatiquement la **normal map** et l'**ambient occlusion map**. On a déjà parlé de la normal map, l'AO map est la suivante sur la liste.

L'**ambient occlusion** consiste, en deux mots, à assombrir les recoins d'un objet. Empilez des assiettes et regardez bien l'espace entre elles : la lumière ambiente ne parvient pas aussi facilement à éclairer cet espace fermé. L'éclairage des jeux vidéo ne suffit pas à produire un résultat pareil, il faut là-encore tricher. On va donc générer une **ambient occlusion map** en même temps que la **normal map**.

Pour réaliser le **baking**, vous devez superposer le low poly et le high poly dans la même scène. Aucune caméra ou light n'est nécessaire ou même souhaitable, nous n'allons pas procéder à un rendu classique. Le low poly doit être associé à une texture : en mode *edit*, sélectionnez tout, puis dans la fenêtre d'édition UV, sélectionnez également toutes les faces si elles ne sont déjà. Créez une nouvelle texture, de préférence à une résolution doublée par rapport à votre taille cible.

i

Aujourd'hui, une texture typique a une résolution de 1024 pixels de côté, mais il est très courant d'utiliser 2048 pixels, voire 4096 pour de très gros objets lourdement détaillés (personnages, véhicules...). Un baking à une résolution double permet d'avoir un rendu plus fin, plus précis, avec moins d'artefacts ; en revanche votre système peut être mené à rude épreuve. Un baking d'ambient occlusion en 8192 pixels, à moins de disposer d'une machine surpuissante, peut durer plus d'une heure...

Pour lancer le baking, il faut se placer dans l'onglet **Render** du panneau latéral, puis de dérouler la section **Bake** tout en bas. Dans **Bake Mode**, il faut choisir **Normals** ou **Ambient Occlusion** suivant la texture à produire, cocher **Selected to Active** et configurer les propriétés **Distance** et **Bias** si elles existent :

— **Distance** configure la distance maximale entre le high poly et le low poly, pour une face ;
— **Bias** indique la distance de laquelle le high poly peut dépasser du low poly.

Il suffit ensuite de sélectionner le high poly puis le low poly avec la touche *shift*, et de cliquer sur **Bake**.

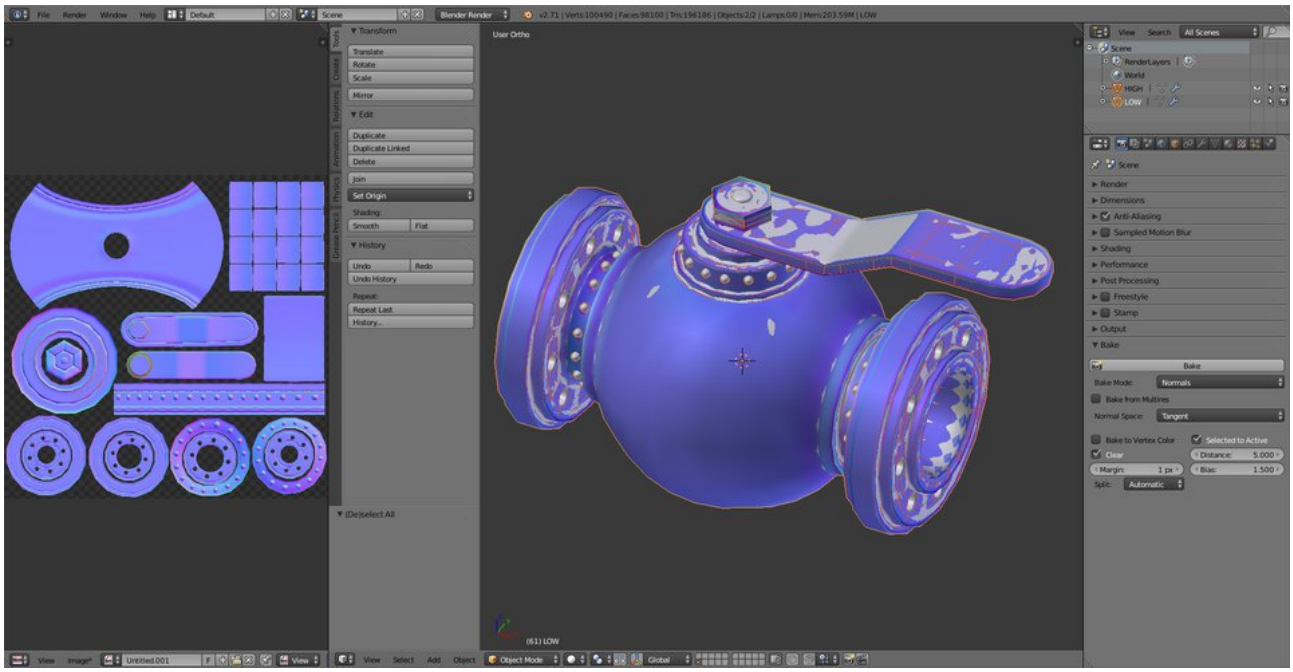

FIGURE 6. – Le baking, tout juste terminé : on voit que le high et le low se superposent

L'opération doit être faite pour les deux types de texture, *via* le menu déroulant. N'oubliez pas de sauvegarder une copie de chaque texture. Si vous avez fait le baking à une résolution plus élevée que la résolution cible, c'est aussi le moment de les redimensionner dans votre éditeur d'image favori.

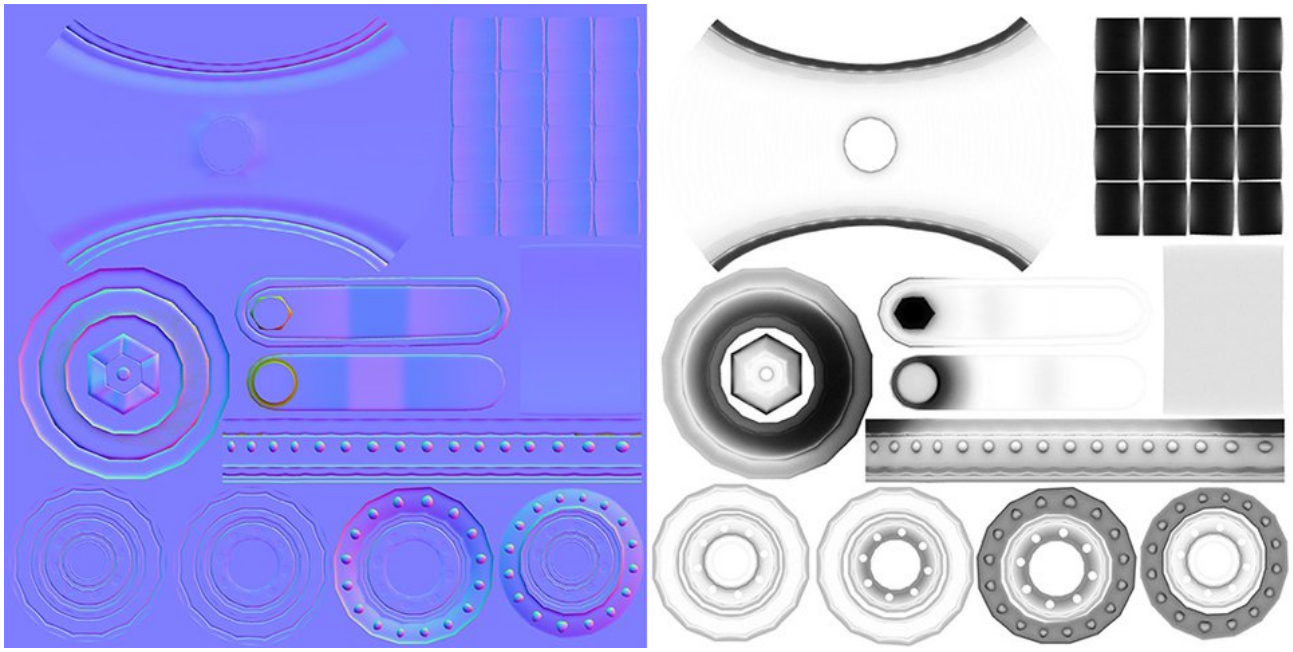

FIGURE 6. – Normal map et ambient occlusion map

7. Albedo, roughness, metalness : la surface dans tous ses états

Au terme de ce travail, vous disposez de la moitié des textures à importer, ainsi que d'un mesh low poly. Il nous manque encore deux textures, et elles vont toutes deux nécessiter un travail manuel dans un éditeur d'image.

On va donc s'attaquer à la texture la plus classique de toutes, l'**albedo**. Elle fut connue à une autre époque comme la **diffuse map**, et c'est aussi celle qu'on appelait avant **la texture**, tout simplement, car il n'y avait qu'elle. Elle représente la couleur, et uniquement la couleur de l'objet.

Le travail à faire ici dépend entièrement de l'objet, des outils, de la direction artistique du jeu. Habituellement, on utilise une texture dite de base, comme du métal ou du bois, en dessinant par dessus les détails qui rendent l'objet unique : éraflures, peinture, marquages...

i

Dans le cas de l'objet d'exemple, voici ce que j'ai fait :
- j'utilise une *base texture* de métal ;
- ajoute de la rouille dans les recoins ;
- je peins ensuite, à l'aide d'une tablette graphique, au pinceau sur toutes les arêtes dans une couleur plus claire afin de mettre en valeur les angles ;
- j'ajoute des touches de couleur pour représenter soit du cuivre, soit de la peinture sur l'objet ;
- enfin, j'ai dessiné une étiquette sur la manette de la valve.

Un objet métallique est relativement simple à faire, ça se corse si vous devez réaliser un personnage, bien sûr.

La **roughness map**, dernière de la liste, va exiger beaucoup moins de talent artistique. Elle représente l'état de surface de votre objet : le noir représente une surface plate comme un miroir, le blanc représente du ciment ou du papier de verre. En toute logique, du métal propre va avoir une *roughness* très faible, alors que du métal rayé va avoir une *roughness* supérieure. C'est tout !

En pratique, une roughness map de qualité va souvent être obtenue à partir de la *base texture*, en augmentant le contraste et en dessinant les arêtes de façon très visible.

?

C'est bon, c'est fini ?

Pas tout à fait : on ne s'est pas encore occupés de la **metalness**. On rappelle que ce paramètre définit si le matériau se comporte comme un métal :

- à 0, l'objet est dit *diélectrique* (comprendre *isolant*), ce n'est donc pas un métal ;
- à 1, l'objet est parfaitement métallique ;
- logiquement, une valeur de 0.75 par exemple serait adaptée pour un métal un peu sale.

Ce réglage va principalement jouer sur la couleur et l'intensité des réflexions sur votre objet. A vous de jongler avec ce paramètre pour réaliser le matériau parfait, j'ai choisi d'utiliser la valeur 1 pour cet objet.

8. C'est prêt

Vous disposez maintenant d'un objet complet, prêt à être importé dans votre moteur de jeu favori. Il reste donc cette dernière étape, pour laquelle je ne peux pas vous aider : chaque jeu est différent, chaque technologie a ses propres méthodes. On sort du domaine de la méthode pour rentrer dans des problèmes d'organisation.

D'une façon générale, le principe est le même partout. Il faut importer l'objet FBX, les textures au format TGA ou PNG, créer un matériau à partir de ces textures, et l'associer à votre objet importé.

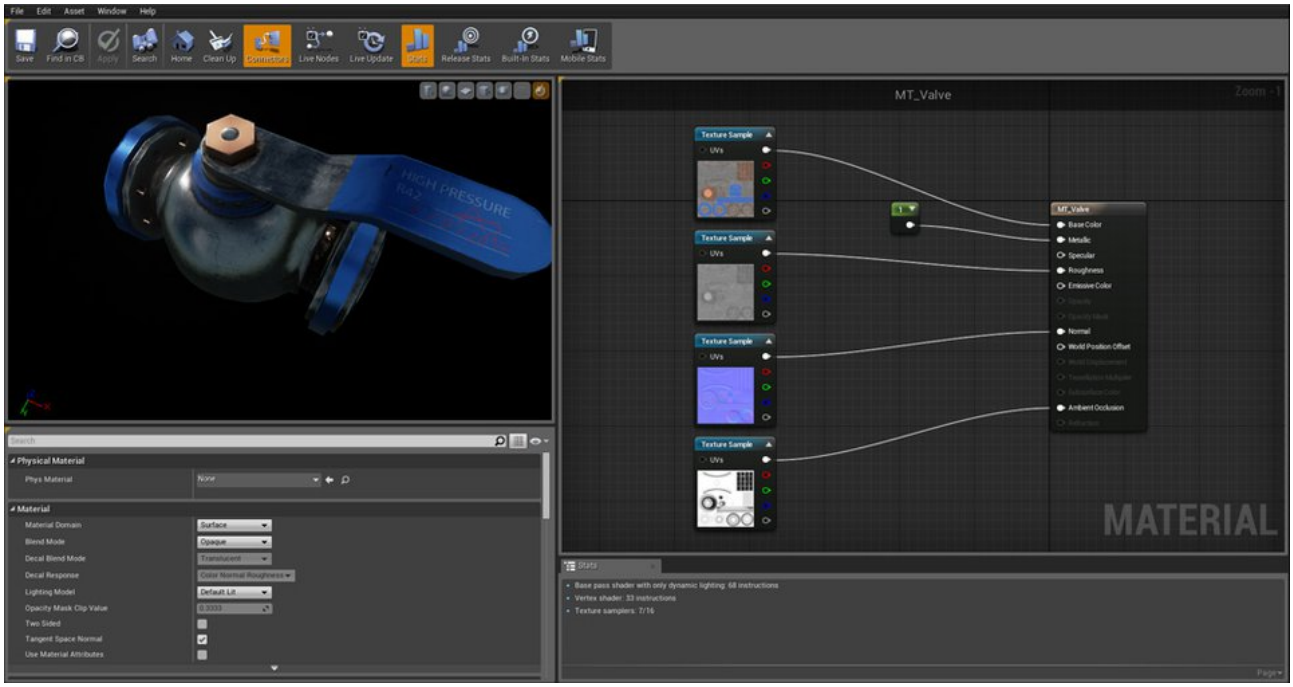

FIGURE 8. – Le matériau dans Unreal Engine 4, après quelques réglages

Chaque moteur de jeu a ses propres caractéristiques de jeu et il faut bien sûr être conscient qu'un objet va toujours exiger des réglages particuliers : textures plus lumineuses, plus contrastées... Les images ci-dessous montrent des résultats différents dans des outils différents.

FIGURE 8. – Le rendu dans Unreal Engine 4

FIGURE 8. – Le rendu dans Marmoset Toolbag

Au terme de ce cours, vous avez en main les principales notions pour réaliser des objets dans un jeu vidéo.

Les fichiers utilisés ici comme exemples sont disponibles au téléchargement. La licence qui s'y applique est la même que celle indiquée pour ce cours.

8. C'est prêt

— fichier source Blender
— fichier source Photoshop
— archive compressée des ressources au rendu

Merci de votre lecture !