LES OUTILS DU JEU VIDEO, SUPPORTS D'UNE INTENTION GRAPHIQUE POUR LE RENDU D'ARCHITECTURE

RAVRY Jérémiah
Master Design Global - A.M.E, Architecture, Modélisation, Environnement

Mémoire de Fin d'Etudes
Structure d'accueil : CRAI (UMR MAP CNRS), ENSANancy
Directeur de mémoire : Damien HANSER

Ecole Nationale Supérieure d'Architecture de Nancy
Soutenance le 11 septembre 2017

AVANT PROPOS

L'intérêt pour la visualisation et l'image s'est développé durant mes années d'études à l'école d'architecture d'une part, mais aussi de part ma culture personnelle et ma pratique du jeu vidéo.

L'image n'est pas dénuée de signification et le jeu vidéo comme industrie a la capacité de générer les environnements permettant de répondre aux questions du joueur, tout en assumant un style graphique.

Dès lors, j'ai cherché à comprendre comment ces outils pourraient être exportés pour l'architecture et en quoi ces systèmes pourraient permettre de nouvelles manières de communiquer le projet à plusieurs types de publics.

REMERCIEMENTS

Je tiens à remercier mon amie Lucile Courteaux, pour son aide sur la fin de l'écriture de ce mémoire, la finalisation de la mise en page et son avis plein de recul sur mes pensées.

Je tiens également à remercier Damien Hanser, pour son suivi et son encadrement tout au long du processus de réflexion effectué pendant un stage dans le laboratoire du CRAI.

Pour finir, un grand merci à Elodie Hochscheid pour ses conseils, sa ténacité et son acharnement à vouloir aider, même lorsqu'elle n'y est pas obligée.

SOMMAIRE

INTRODUCTION

Après de nombreuses réflexions, le sujet de ce mémoire s'est porté sur le sens de l'image et du rendu d'architecture ainsi que le jeu vidéo comme support technique de travail à la mise en service des intentions graphiques du rendu. Ce choix a été motivé par mon intérêt pour la visualisation architecturale et ma pratique du jeu vidéo.

En tant qu'industrie, le jeu vidéo a une capacité de travail importante et possède les outils dédiés à leurs représentation graphiques. Les jeux ne se limitent pas à des mondes réalistes, mais explorent plusieurs styles, plusieurs univers, plusieurs ambiances, sans toutefois varier les outils de base. Dès lors, nous pourrions nous demander pourquoi ces outils ne sont pas plus utilisés dans le milieu de l'architecture alors qu'ils offrent des moyens de développer la visualisation du projets de différentes façons.

Pour répondre à cette question, le plan est constitué des trois parties suivantes : une représentation sémantique, les outils issus du jeu vidéo et l'application aux ambiances.

Tout d'abord, nous chercherons à comprendre en quoi l'image et l'environnement sont tous les deux chargés de sens et pourquoi la question de le représentation est une question essentielle dans la communication du projet, mais aussi dans bien d'autres domaines. Le rendu d'image en architecture cherche à retranscrire une volonté de l'architecte, dans une recherche d'intégration du projet dans un contexte, tout en lui donnant sa propre identité. Il s'agit alors de voir en quoi le moteur de jeu et les nouvelles technologies numériques peuvent permettre une évolution de cette représentation et une diversité des ambiances au sein d'un même modèle.

Puis, nous allons nous intéresser au jeu vidéo et ses outils, afin de comprendre les contraintes techniques et matérielles permettant la création d'univers différents. Une analyse technique permettra de mettre en avant l'importance du travail de modélisation en amont ainsi que la préparation au travail de la texture, par une méthode appelée l'UV Mapping. Nous allons ensuite étudier les nouvelles technologies d'affichage et de texture, notamment le PBR, permettant d'utiliser différents types de textures pour différents usages, en générant des effets de lumières et de relief.

Ensuite, nous développerons une approche par des outils gratuits pour développer un flux de travail permettant de développer le PBR et la texture à des fins de visualisation architecturale pour un moteur et permettre la création d'une scène, qui sera éprouvée de différentes manières.

Enfin, l'étude d'un moteur de jeu, Unity 3D 5, nous permettra de mettre en avant les qualités du jeu vidéo à l'ère numérique, avec le développements d'outils permettant une grande liberté créative au service de plusieurs visualisations. En effet, la simplicité d'utilisation offre de grands possibilités, qu'elles soient graphiques ou physiques. Nous terminerons sur l'ajout récent des technologies de réalité virtuelle ainsi que les opportunités qu'elles offrent au rendu en temps réel et à l'appropriation d'une scène.

Ce plan permettra d'aborder les questions théoriques des bases de l'image, les règles de composition, mais aussi les questions techniques et comment les méthodes du jeu vidéo peuvent être considérées pour la visualisation architecturale.

PARTIE 1 :
UNE REPRÉSENTATION SÉMANTIQUE

A Histoire de la représentation

Définition

Le rendu d'architecture est une représentation du projet dont le but est de mettre en avant ses qualités, intégré dans un contexte. Pourtant, avant de réaliser l'image, il faut débuter un projet, le dessiner, et en définir les intentions qui permettront d'affirmer son identité par l'image.

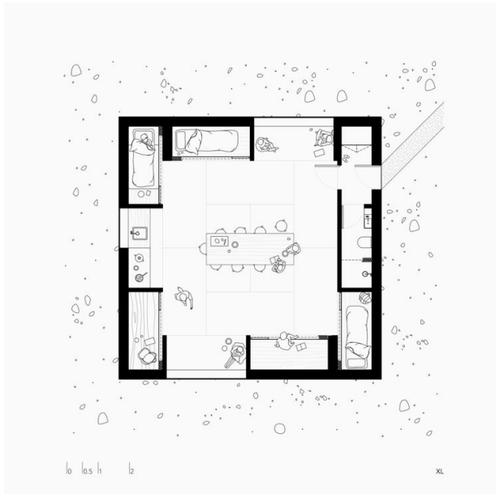
Un projet est représenté par un ensemble de dessins assurant sa compréhension aux personnes formées à les interpréter. Un dessin peut être de plusieurs types, du croquis au détail, mais tous représentent une volonté géométrique, techniques et esthétiques de l'architecte, le tout étant cadré par des conventions techniques (figures 1 à 4).

Nous pouvons considérer que le rendu tel que nous le connaissons actuellement est apparu avec les premières notions de la perspective, à la Renaissance italienne.
Il convient, à ce stade, de faire la différence entre le rendu et le dessin. Le dessin est défini par une « **Représentation sur une surface de la forme (et éventuellement des valeurs de lumière et d'ombre) d'un objet ou d'une figure, plutôt que de leur couleur.** »[1].

La caractéristique principale est alors de montrer l'objet dans une vision simple, mettant en avant les formes plutôt que l'aspect plastique et graphique.

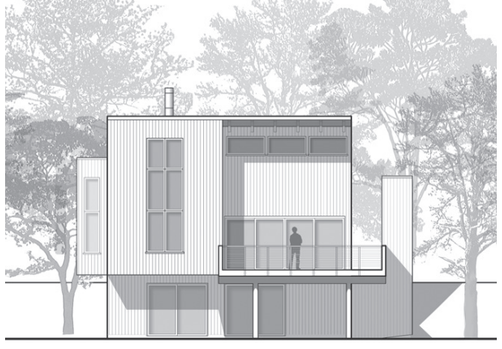
Un rendu, par opposition est une représentation des formes avec une recherche plastique et graphique d'un objet, dans le but de le mettre en avant de manière plus significative.

1. Définition du Larousse
http://www.larousse.fr/dictionnaires/francais/dessin/24654



1. Représentation graphique d'un plan

2. Représentation graphique d'une coupe



3. Représentation graphique d'une élévation

4. Perspective de projet

Débuts de la perspective

Précédant cette période, au Moyen-Âge, les éléments picturaux ne présentent pas de règles de profondeur ou d'espace, les différents personnages et objets sont représentés par leur importance dans la scène, sans réalité de volume (figure 5).

L'un des précurseurs de la perspective est Giotto, qui utilise des volumes simples dans ses représentations, souvent des boites, qui vont lui permettre d'accentuer l'effet de profondeur en y insérant les personnages (figure 6).

Toutefois, c'est l'œuvre d'un architecte à la Renaissance qui va définir les règles de constructions de la perspective actuelle. En 1415, Brunelleschi peint un petit tableau représentant le Baptistère de Florence et sa place, en utilisant un panneau de bois percé et un miroir pour vérifier la justesse de la perspective. Le mot « perspective » nous provient du latin « perspectus », signifiant « regarder à travers, regarder attentivement».

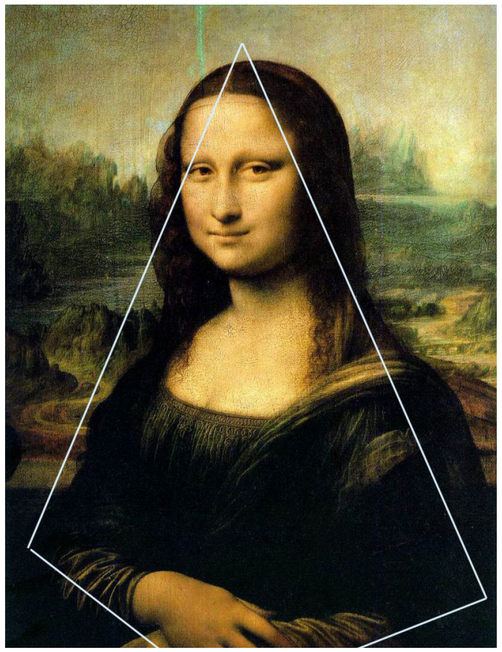
Le travail de Brunelleschi est repris dans un très grand nombre d'œuvres, peu de temps après ses premiers travaux, nous pouvons notamment citer « la cité idéale » de Piero della Francesca (figure 7).
La formalisation des règles de la perspective au service de l'art se sont affinés et se retrouvent dans toutes les grandes œuvres et dessins, de cette période à notre époque (figure 8 & 9).

Le contexte social de la renaissance a remis en avant l'être humain, et le pouvoir en place reprends le contrôle face à l'église. Le changement du mode de pensée va permettre des explorations dans les domaines scientifiques et artistiques, en abandonnant l'aspect explicatif et pictural de l'imagerie religieuse.

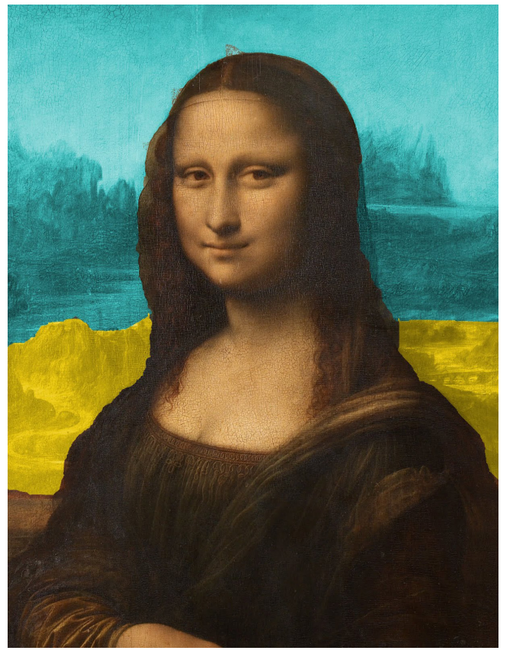
5. Madonna and Child

6. Annonciation à Sainte-Anne

7. Cité idéale



8. Tracés régulateurs



9. Profondeur

Les recherches scientifiques menées par les savants de cette époque sur l'homme et la nature ont eu de grosses influences sur les différents courants se succédant après la renaissance. Dans le milieu du 17ème siècle, deux courants s'opposent dans leurs intentions de représentation, l'art baroque et le classicisme.

Les mouvements artistiques

Le terme « Baroque » nous vient du portugais « barroco », utilisé dans le vocabulaire de la joaillerie pour définir une perle présentant des irrégularités. Ce mot est lui-même dérivé du latin « verruca » pouvant se traduire par « verrue » ou « défaut ».

Ce courant a évolué entre 1600 et 1720 environ, mais n'a été nommé de cette manière qu'au XIXe siècle. La façon de nommer ce courant n'est évidemment pas anodine, et renvoie à l'approche effectuée par les artistes de cette période.
En effet, l'art Baroque ne tranche pas directement avec les représentations de la Renaissance, mais cherchent à évoluer dans la mise en scène, pour présenter des œuvres portant un cadre fort, au-delà de la réalité formelle d'une scène.

Les règles de composition sont transformées pour privilégier la diagonale, l'éclairage est différent et peut changer d'un endroit à l'autre de la scène, les personnages sont en action et sont les représentants de l'émotion de la scène, qui n'est plus véhiculé par des messages du décor (figure 10).

Le classicisme s'installe en opposition à ce mouvement, en voulant renouer avec la tradition picturale de la Renaissance. L'un des artistes forts de ce courant est Annibal Carrache, dont les travaux recherchent une composition claire, qui fournit toutes les clés au spectateur pour comprendre le message. Il n'y a donc pas d'éléments soumis à interprétations, et la scène est très souvent comprise dans le cadre, ce qui permet de centrer le message et l'action.
Ce courant a été le fer de lance de la monarchie absolue sous Louis XIII et Louis XIV, afin de présenter le roi sous son plus bel aspect, afin de maintenir sa supériorité sur l'Europe (figure 11).

Entre 1725 et 1789 avec la révolution française s'installe le style Rococo. En lien avec la réalité dans laquelle ils évoluent, les artistes s'éloignent du monde en dépeignant des scènes travaillées, recherchant à représenter le monde tel que nous le souhaitons, plutôt que tel qu'il devrait être.
Nous retrouvons alors des formes plus libres, déjà expérimentés dans l'art Baroque, ainsi que des éléments retirés de la Renaissance, notamment les symboles et ornements au service de l'illusion de la scène.

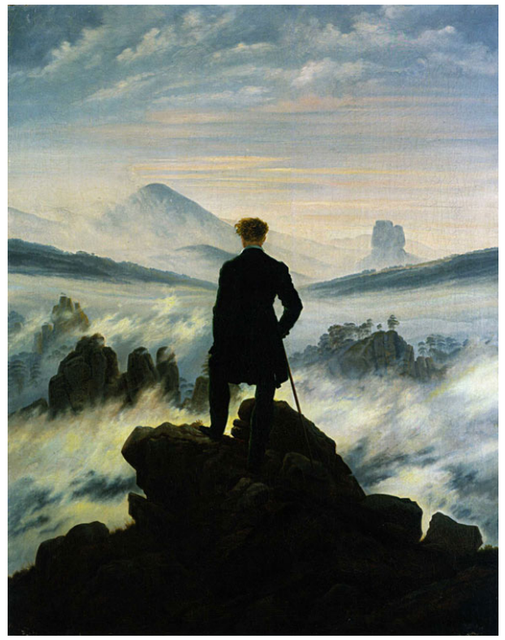
Le rococo est une recherche de l'illusion à travers la représentation, dans le but s'échapper du réel, pour masquer ses émotions face à la réalité de l'époque et son contexte social. En prenant l'exact contre-pieds apparaît le mouvement romantique (figure 12).

Encore une fois, les oppositions se creusent par les moyens de représentations, et permettent de suivre l'évolution du contexte social aux travers des âges. L'art romantique est un langage artistique reflétant les sentiments et pensées de l'artiste, le rêve, la peur, le doute au travers du prisme de la réalité (figure 13).

10. La laitière

12. Vue imaginaire de Rome

11. Portrait de Louis XIV



13. Le promeneur

Après la révolution française et au moment de la révolution industrielle, l'art se cherche, entre la monarchie et la république, cette période d'incertitude et de changement est le catalyseur de cet art.
A partir de 1850, l'art évolue pour exploser en un grand nombre de courants, avec l'évolution des moyens de propagation des savoirs dans le monde.

Sous le second empire, l'art délaisse la mythologie et l'histoire pour s'intéresser à la vie quotidienne de la France, alors en pleine évolution, mais toujours coincée entre les traditions, évolutions sociale et révolution industrielle. Le courant Réaliste est majoritairement relié au travail de Gustave Courbet, qui cherche à se détacher un peu du romantisme pour se concentrer sur l'homme en tant qu'être humain dans un contexte social réel, en opposition au romantisme et son imaginaire (figure 14).

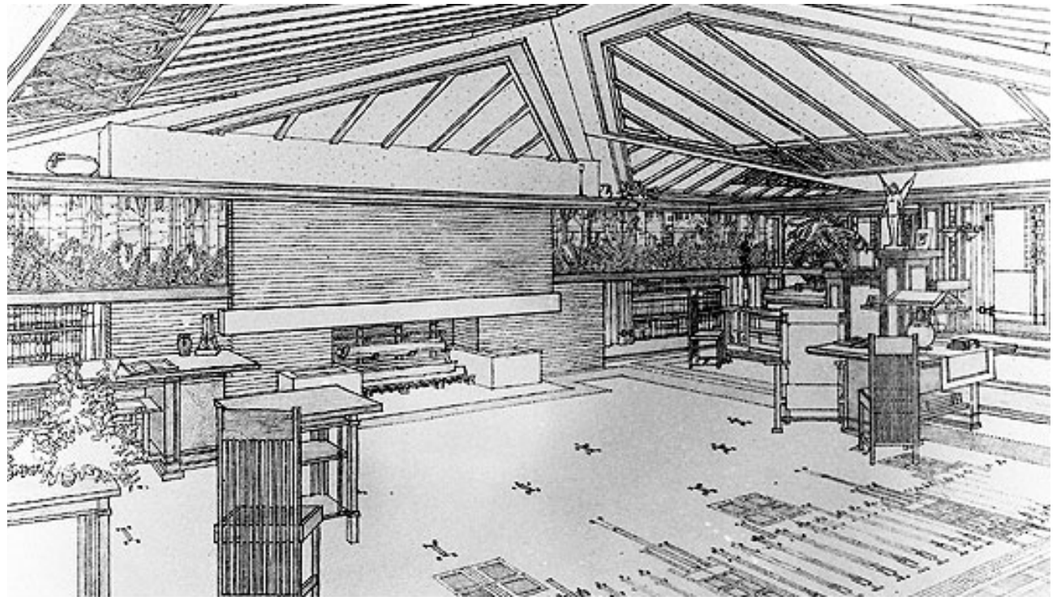
Le courant impressionniste, à la même époque, peut être considéré comme une extension du réalisme, dans la peinture de scènes de la vie quotidienne, mais en y travaillant une composition d'image tenant compte de lumière et des effets sur la matière, mais reflétant une impression plutôt qu'un moment réel (figure 15).

L'époque contemporaine est l'époque de naissance d'un grand nombre de mouvements artistiques abstraits, nous intéressant moins dans la représentation du projet.
Les courants évoqués auparavant ont tous un objectif de représentation bien précis mais diffèrent aussi par leurs moyens de représentations et la manière d'approcher une scène.
Toute cette culture picturale est encore réinterprétée aujourd'hui en architecture, au cinéma, dans le jeu vidéo, selon le message que l'artiste cherche à faire passer, et l'élément qu'il cherche à représenter (figure 16).

Pourtant, l'élément marquant du rendu tel que nous allons le voir a été l'apparition des premiers systèmes informatiques permettant le dessin.

14. Autoportrait

15. Les raboteurs de parquets



16. Dessin de Frank Lloyd Wright

Dessin assisté par ordinateur

Nous pouvons définir l'origine du dessin assisté par ordinateur (DAO) en 1962 avec l'outil Sketchpad, qui permet la création de formes géométriques simples, qui peuvent être utilisées comme groupes pour ainsi partager les propriétés des objets. Chaque forme pouvait être modifiée élément par élément, de la longueur du trait, à la taille d'un angle (figure 16).

Cette invention est à mettre en parallèle avec un second élément très important pour l'architecte, la diffusion et la présentation de ses plans.

En 1964 apparait la première imprimante à jet d'encre de l'histoire, la ***Teletype Inktronic***. Il faut toutefois préciser que cette imprimante ne permettait alors que d'imprimer des caractères de type ASCII, et non des images de pixels ou de vecteurs, limitant l'utilisation à certains secteurs.

A ce moment, les méthodes de représentations par ordinateur ne sont pas encore assez développées et ouvertes pour permettre l'utilisation par les corps de métiers non spécialisés, et encore moins par un public profane. Ces outils sont très majoritairement utilisés par l'industrie, notamment automobile et aéronautique.

La dizaine d'années suivante est connue pour la médiatisation des premiers ordinateurs individuels, au milieu des années 70, avec la création de ***Micro-Soft*** que nous connaissons maintenant sous le nom évident de ***Microsoft*** mais aussi l'apparition des premiers ordinateurs de la marque ***Apple***.

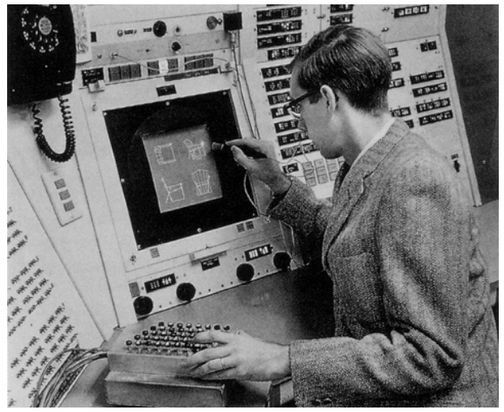
La propagation de ces nouveaux outils permet aussi la création des logiciels adéquats, désormais accessible à un plus grand nombre de personnes.

En 1980, François Postaure effectue la première esquisse numérique en tant que rendu de fin d'étude, à l'école d'architecture de Normandie, en utilisant notamment un ordinateur Apple 2 (figure 18).

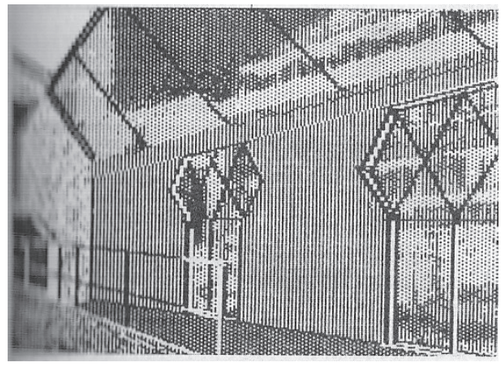
La représentation figurative au trait a rapidement évolué en même temps que les technologies pour les industries du divertissement, pour permettre l'apparition de différents outils adaptés au rendu tels que Rhino, VRay, Maxwell, Mental Ray ou encore Artlantis.

La construction du projet sur ordinateur a aussi évolué, en passant de la 2D, à la 3D, avec des outils simples tels que sketchup ou Rhinoceros, pour arriver aujourd'hui vers le BIM, se traduisant par ***Building Information Model***, c'est-à-dire une construction du projet en trois dimensions, par des objets, murs, fenêtres, portes, toitures, avec une surcouche d'informations en plus, l'épaisseur du mur, sa composition, sa surface, permettant de récupérer automatiquement des données de gestion d'entretien, de calculs thermiques, etc.

L'image de rendu en tant que tel s'est démocratisé à notre époque, et est présente partout, de la publicité au cinéma, en passant évidemment par l'architecture. Dans une société consumériste où l'on chercher à vendre un produit, l'image est fondamentale, et cela pose question à l'architecte. Nous allons donc voir le rôle ainsi que la création du projet au sein d'une agence pour comprendre sa place dans le développement d'un projet.



17. Utilisation de Sketchpad



18. Esquisse numérique

Outils de communication

En agence, le rendu est un outil important de communication, à tous niveaux. L'objectif est de rester dans le milieu d'une agence d'architecture, avec le contrôle de l'architecte de la phase de conception à la réalisation, et donc ses propres intentions formulées au travers, non seulement du projet, mais aussi des documents graphiques l'accompagnant.

Le rendu en tant qu'outil de visualisation a un rôle très important dans le projet, et ce, dès les premières phases de conception. Il est même un outil nécessaire de communication, entre plusieurs personnes au sein d'une agence, avec des entreprises mais aussi des clients.

L'image est un vecteur de communication très fort, pour faire passer ses idées et intentions, cela se vérifie tous les jours avec le monde qui nous entoure, des affiches publicitaires au produits que nous achetons, l'image est indissociable de la représentation du monde, seul la manière de la créer évolue (figure 19).

L'architecture se différencie par la complexité de l'objet à représenter et son médium, une image en deux dimensions. L'architecte dessine un espace et un volume qui sera expérimenté, vécu et habité par les clients ou usagers du futur lieu. La notion d'espace est ici très importante, et doit être respectée pour faire comprendre au spectateur les intentions au-delà de l'image [1].

Le rendu et la visualisation d'un projet peut être à double tranchant, très utile ou contre-productif dans différentes situations, la phase de conception et la présentation.
La plus grande force du rendu est sa capacité à communiquer et être accessible à un public ne pouvant pas lire les plans correctement.

En phase de conception il peut être utile pour montrer à un client une intention ou stimuler le processus créatif par la découverte de qualités d'espaces amenant à de nouvelles problématiques de développements de projet.

A l'opposé, le rendu peut faire perdre le concept du projet dans le cas de détails trop fournis [2], d'une recherche d'un scénario trop dirigé, ne laissant aucune liberté à l'usager pour s'imaginer dans l'espace et se projeter dans le volume dessiné.

Pour autant, l'image peut aussi être enjolivée pour faire accepter un projet qui pourrait faire poser question s'il avait été représenté autrement. La supercherie visuelle du rendu est un questionnement actuel du milieu de l'architecture, avec l'expansion des méthodes de rendu et une finesse technique de plus en plus poussés dont les architectes perdent le contrôle (figure 20).

L'agence de Peter Zumthor a, par exemple, proscrit les rendus de ses concours [3] et présentent uniquement des maquettes des projets.

1. The importance of visualisations in architecture / Andrew Brett / 7 Juillet 2013

2. Are 3D renderings deceiving architects and clients / John Kutyla / Ocotbre 2015
Lien : http://www.archdaily.com/774853/are-3d-renderings-de-ceiving-architects-and-clients
3. Les rendus sont-ils mauvais pour l'architecture / Nicolas Richelet / 6 Avril 2015

19. Publicité

20. Rendu de projet d'une usine de déchets / BIG

La question de la pertinence de la représentation par la maquette peut aussi être posée, par les matériaux utilisés, l'échelle, etc. mais cela sort du cadre de notre propos.
Il s'agit donc de faire la distinction entre le rendu en tant qu'outil et en tant qu'art, afin de l'utiliser à bon escient, avec le niveau d'échelle et de détail approprié.

Travail en agence

Chaque agence a sa manière de fonctionner, mais le rendu final est le même medium pour toutes, une image. La question de la compétence de production de l'image est liée à la manière de travailler en amont, et le nombre d'étapes entreprises pour arriver à l'image.
En effet, encore un grand nombre de petites agences d'architectures travaillent sur des outils 2D, tels que AutoCAD, et passent ensuite par une phase de modélisation, de rendu, et enfin de post-traitement, sur un logiciel différent par étapes. Cette rigidité d'organisation s'explique par un grand nombre de facteurs dont je ne développerai que quelques points ici.

D'une part, la plupart des chefs d'agences ont été diplômés avant l'apparition des premiers ordinateurs individuels, ce qui explique la difficulté d'adaptation à des logiciels récents, de type CAO. Nous pourrions mettre en avant le questionnement autour des outils actuels de type BIM ainsi que la perte de contrôle du projet dont les architectes ont peur, qui fait encore débat en ce moment.

D'autre part, les architectes n'ont que très rarement le temps et les moyens de s'offrir une formation sur de nouveaux outils, et restent très souvent coincés dans les mêmes mécaniques de travail, même avec des nouveaux outils. Par exemple, certains architectes utilisent un logiciel type BIM tel qu'ArchiCAD pour dessiner en 2D tout comme le fait AutoCAD.

La culture d'une architecture à concevoir en trois dimensions est très difficilement acquise pour toute une génération d'architectes, souvent préférant garder le contrôle sur son travail, par peur d'essayer des logiciels dont les concepts ne sont pas acquis et nouveau.
Il est alors très difficile de faire changer la manière de fonctionner une agence, surtout lorsqu'elle est prise dans un cycle de travail très long d'un projet, qui est de quelques années.
Il s'agit maintenant de voir comment les récentes formations dans les écoles d'architecture ont poussé la conception avec l'usage de la 3D, et ses conséquences sur le rendu.

C L'origine d'une culture

L'article de Oliver Wainwright intitulé ***Why architectural education in Britain is in need of repair*** ? dans les colonnes de ***The Guardian*** nous montre un exemple de réaction à l'apprentissage du métier d'architecte de cette décennie.

Les projets de fin d'études des étudiants des meilleures écoles d'architecture au Royaume-Uni présentent une forte démonstration de compétences graphiques et de modélisation, mais concept [1]. Les projets donnent l'impression de fuir le monde réel, les usagers, le contexte et l'échelle, en imaginant un monde fantaisiste dans lequel le projet n'est présent que par ses formes.

Le plus gros problème énoncé est le manque de concept masqué par une forte complexité visuelle, crée de toute parts grâce aux outils numériques paramétriques, sans même se souvenir du point de départ du projet.
Le cas est intéressant à soulever car les écoles d'architecture au Royaume-Uni sont privées et les frais d'inscription très élevés, ce qui les obligerait à remettre en cause leur modèle d'enseignement régulièrement afin de s'adapter à un contexte du métier d'architecte en constante évolution.

Les outils actuels permettent une simplicité de modélisation et de production de documents du projet, notamment avec les outils de type BIM. Sans culture constructive préalable, il est parfois facile de céder aux facilités de dessin sans réaliser les implications de chaque décision.

Par exemple, l'enseignement d'ArchiCAD dans cette école est un point positif pour s'inscrire dans l'évolution du métier d'architecte, mais il faut attendre quelques temps pour être capable de prendre du recul, et se rendre compte du potentiel ainsi que du rôle de l'outil. En effet, lorsque l'on découvre l'outil, l'enthousiasme de pouvoir créer un projet et en sortir automatiquement tous les documents, créer ses premiers rendus, prends le pas sur l'objectif de chacun de ces documents.
Le constat est le même avec les logiciels de rendu, tels que Artlantis, qui simplifient au maximum les paramètres en fournissant un logiciel très simple d'utilisation, donnant une apparente simplicité de rendu, mais sans grande optimisation et possibilités.
Dès lors, nous pouvons créer des images très simplement et peut-être même trop simplement.

Outre la facilité d'utilisation de certains outils, il faut remettre en question l'enseignement lui-même pour comprendre pourquoi les outils ne sont pas utilisés correctement.
Les enseignants formant les élèves sont très souvent les architectes qui essaient de transmettre leurs connaissances avec les méthodes de travail hérités d'une époque sans les outils de CAO. Indirectement, les problèmes rencontrés au sein des agences se retrouvent dans la manière dont est géré l'apprentissage aux étudiants.

Dès lors, l'outil est très souvent écarté par les architectes alors qu'il est même nécessaire car l'image est le plus puissant des vecteurs de communication pour transmettre des intentions et une vision du projet. Malgré tous les questionnements qu'ils posent, les rendus et l'image sont des transmetteurs d'informations essentiels au métier, et la technique nécessaire à leur élaboration doit être maitrisée pour gérer son image afin de produire un message juste et voulu.

C'est ici que la distinction entre le rendu en tant qu'outil ou en tant qu'art prend tout son sens, car les deux ont un sens et une signification selon l'usage auquel il est destiné, et nous allons voir en quoi un rendu peut prendre de multiples formes avec de multiples messages.

1. Towering Folly – Why architectural education in britain is in need of repair / Oliver Wainwright / 30 Mai 2013

D La photographie virtuelle

Introduction

L'outil est perçu comme un obstacle pour la plupart des architectes, de par la complexité et la technicité pour obtenir une image jugée « belle ». Mais outre les performances techniques des outils de rendu, l'image se définit par plusieurs autres aspects sans doute bien plus importants, auquel ils sont pourtant sensibilisés.

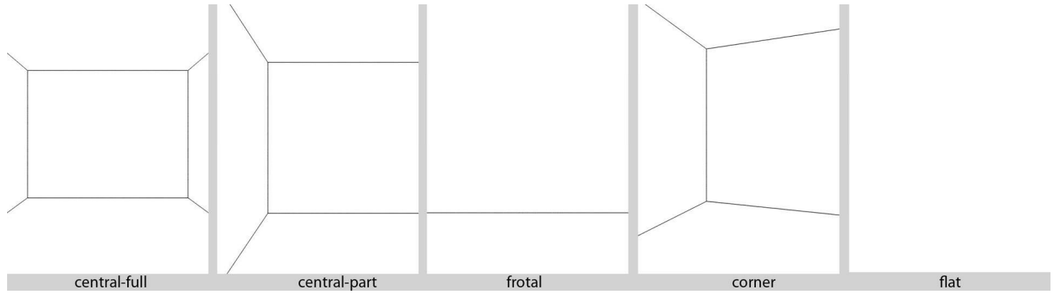
La difficulté du rendu est de créer une représentation du projet qui n'existe pas encore, et donc d'imaginer et créer une vision qui soit assez précise pour créer un espace cohérent et réaliste. L'un des risques est de créer une image pouvant créer des attentes irréelles chez le spectateur, et provoquer une frustration entre lui et l'architecte.

Pourtant l'image de rendu peut être interprétée comme une photographie d'un espace virtuel, et comme toutes les photographies [1], elle est assujettie à des règles déjà connues depuis les premières représentations de la renaissance.

Dès lors, il faut prendre en compte certains points pour construire une image correcte, la hauteur de la caméra, des lignes verticales, une focale proche de l'œil humain ou encore un les règles de cadrage par le système de tiers ou les proportions par le rectangle d'or.

Toute les influences culturelles du rendu viennent de la photographie au sens large, nous pouvons alors établir une distinction de plusieurs grands artistes photographiques afin de dégager des aspects de la photographie d'architecture.

1. Photographic Approach in architectural vizualisation / Lasse Rode – Studio XOXO / 26 Mars 2013



21. Exemples de cadrages

22. Règles des thiers

23. Colorimétrie d'une scène

24. Profondeur et composition des plans

Bérénice Abbott

Comme point de départ, nous choisirons Bérénice Abbott, dont les premiers travaux datent de 1929 à New York. Bérénice Abbott est liée au mouvement ***Straight Photography***, symbolisé par l'utilisation de la photographie et sa capacité à enregistrer tous les détails, contrairement à d'autres arts visuels, en autre la peinture.

En 1929, elle arrive à New-York avec comme objectif de capturer ce qui fait l'essence de la ville. Ses travaux sont remarqués, et en 1935 elle est engagée par le ***Federal Art Project*** sur le projet ***Changing New-York*** (figure 25).

Ce projet a été l'occasion pour Bérénice Abbott de photographier la ville de New-York en mettant en avant la vie des habitants dans le contexte architectural et social de l'époque. L'intention étant de faire réaliser aux habitants que le contexte est une conséquence de leur manière d'habiter (la réciproque étant aussi vraie), en affichant l'architecture et ses liens avec les pratiques des usagers (figure 26).

Ces photos sont un premier point de départ d'une approche de l'architecture au début du XXeme siècle, avec l'inventaire d'une vie sociale à l'échelle d'une grande ville.

Bernd et Hilla Becher

Nous poursuivons avec le couple Bernd et Hilla Becher, dont le travail débuté en 1957, est centré sur le bâtiment lui-même.

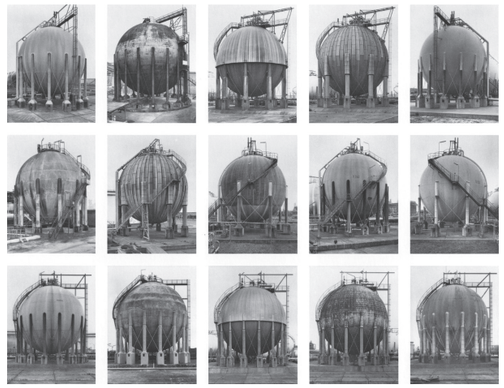
Ce couple de photographes allemand définit son travail par la prise de vue d'ouvrages industriels de manière documentaire et neutre. Chaque série de photographie dédiée à une typologie de bâtiments est effectuée de la même manière, en plaçant le bâtiment au centre de l'image, en l'isolant le plus possible de son contexte et son environnement, d'un point de vue surélevé avec un téléobjectif pour éviter les déformations (figure 27 & 28).

Les groupes de bâtiments sont triés par fonctions et types, tandis que le protocole de prise de vue permet d'ajouter des photos prise plus tard à un groupe sans que la différence ne se fasse sentir. Cette abnégation au protocole de la photographie leur a permis de détailler plus de quarante ans de travail de manière uniforme, tout en remaniant constamment leurs collections et classements [1].

Nous avons là une approche de l'architecture avec un point de vue neutre, dans le cas d'un inventaire du patrimoine industriel. Nous suivrons l'évolution de ce modèle photographique avec Andreas Gursky, qui a été un des élèves de Bernd et Hilla Becher. Mais avant, nous allons voir en quoi le travail de Julius Schulman a été un autre pivot dans la photographie d'architecture.

25. Theoline Pier 11, East River, NY

26. Pike and Henry Street, NY



27. Spherical Gas Tanks

28. Harry E. Colliery Coal Breaker

1. Bernd et Hilla Becher / Centre Pompidou Paris / Octobre – Janvier 2005

Julius Schulman

En opposition au travail du couple Becher, Julius Schulman fixe son attention sur la vie qui se crée par l'architecture. Ses photographies avaient comme objectif d'éclairer les intentions architecturales du projet au travers d'une vision sociale de l'époque. Il a fait partie de la génération de photographes d'architecture d'après-guerre dont l'orientation des images était commerciale, et plus particulièrement orienté vers les projets modernistes, comme la démonstration d'une manière de vivre contemporaine (figure 29).

L'intérêt étant alors de mettre en avant l'époque moderne et l'avancée de la société dans tous les domaines, par des photos noir et blancs, avec comme objectif de réduire le projet à ses plus pures formes géométriques, et faire apparaitre chaleureux les matériaux froids du modernisme tels que l'acier, le verre ou le béton [1] (figure 30).

Son travail le plus connu concerne les séries de photos des maisons californiennes dans les années 50, notamment celles des projets de Frank Lloyd Wright, Richard Neutra, Mies van der Rohe ou Oscar Niemeyer. Nous pouvons le considérer comme le précurseur d'une image pensée pour le projet, porteuse d'idées mais aussi de fantasmes de la vie en Californie (figure 31).

Andreas Gursky

Andreas Gursky a été un élève de Bernd et Hilla Becher à l'école des Beaux-Arts de Düsseldorf en 1980. L'influence de leur travail est plus perceptible dans ses premiers travaux, avant de trouver son thème de prédilection, l'humain au travers du décor.

Le travail de Gursky fait le lien entre deux époques par l'utilisation d'une multitude de techniques et de savoirs, avec l'utilisation de l'appareil photo argentique et du reflex numérique. L'utilisation de ces nouvelles méthodes lui permet de créer des photographies retouchées à des très grands formats, mettant en scène l'homme sous différents thèmes. L'apparition du numérique est le point tournant de sa carrière, avec son panorama ***Paris, Montparnasse*** en 1993 (figure 32).

L'un de ces thèmes forts et la question de l'influence de l'architecture post-moderne de verre et d'acier sur l'individu, le plus souvent exploité par des photographies de bâtiment retouchés et rassemblés pour former une mosaïque de façade, nous interrogeant sur l'individu dans son habitat, la vie et les usages.

Ces compositions ne sont possibles que par la maitrise de la photographie numérique et de ses outils [2], lui permettant d'assembler, créer et composer des scènes même abstraites, amenant vers une nouvelle façon de penser la photographie, non pas comme un instantané d'une scène, mais comme une ressource permettant de créer son propre contenu à partir des images de la réalité (figure 33).

29. Case Study House No. 22

30. Everyday California

31. Solomon R. Guggenheim Museum

33. Circuit automobile de Bahreïn. Compostion abstraite

32. Paris, Montparnasse

1. Julius Shulman, Photographer of Modernist California Architecture, Dies at 98 / NY Times, Andy Grundberg / 17 Juillet 2009

2. Entre monumentalité et petitesse, Andreas Gursky / Karolina Murdzek et Lucile Courteaux / 08 Octobre 2015

Photographes contemporains

Nick Hufton & Allan Crow sont des photographes actuels, basant leur travail sur la photographie numérique. La technologie numérique leur permet de prendre possession d'un lieu ou d'un projet en capturant tous les points de vues présentant un intérêt, en collectant un grand nombre d'images, pour ensuite les combiner, les retravailler et imaginer une histoire que le projet peut raconter (figure 34).

Dès lors, il est beaucoup plus simple de créer un sens à un lieu, en l'imaginant à l'avance et en modifiant légèrement la réalité pour créer une vision du projet.

Iwan Baan exerce dans la même temporalité et concentre sur travail sur le projet en tant qu'espace de vie, approprié ou habité, avec comme intention de communiquer avec les usagers des manières d'utiliser l'espace. L'autre point intéressant est la prise de vue aérienne de projets, plus facile à créer et modifier avec des outils actuels (figure 35).

Pour terminer sur la question de la photographie et de ses composantes, nous pouvons mettre en avant le travail de Ludovic Zacchi, fondateur du collectif « Ilulisä », ancien élève diplomé de l'ENSA Nancy.

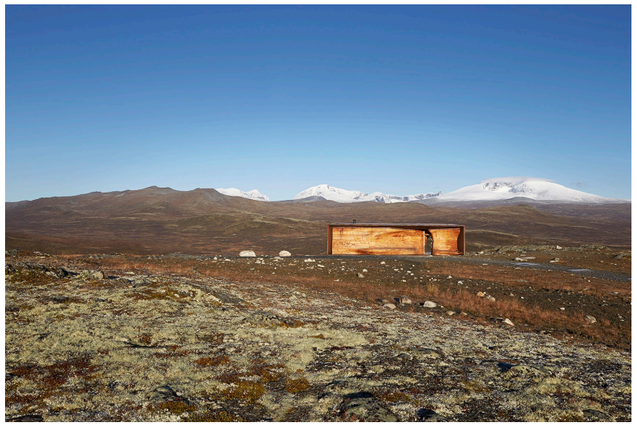
Son travail actuel est centrée sur l'illustration d'architecture pour des agences, avec comme objectif de « mettre en avant le processus de conception mené par les architectes ». L'intérêt de son travail vient de sa formation d'architecte et son expérience dans la maitrise d'ouvrage, lui permettant d'avoir un esprit critique sur un projet, et la vision à en donner avec l'architecte.

Nous pouvons citer ses mots lorsqu'il s'agit de décrire une illustration d'architecture :

« Nous produisons des images comme une peinture, selon le contexte, l'architecture et la temporalité. L'Architecture devient paysage à travers une histoire racontée, une poésie naît en relation avec l'architecte. L'Architecture est insérée dans une scène de vie de tous les jours, elle n'est pas rendue comme un objet isolé déconnecté des hommes et de son environnement. »[1].

Ce qui ressort de la photographie d'architecture, au-delà des règles de composition et de cadrage, est la composante du réel et du contexte, dans lequel un projet s'inscrit et avec lequel il interagit (figure 36).

Le rendu par rapport à la photographie apporte un avantage indéniable, celui de créer des éléments inexistants, de pouvoir se permettre des modes de représentations différents, tout en gardant ce qui fait la force de la photographie. La question de la photographie numérique en est d'ailleurs un exemple marquant, puisque les photographes professionnels ont recours à ces méthodes et cette nouvelle technologie pour reconstituer, recomposer des paysages et des panoramas à partir du réel.



34. Hufton + Crow / Norwegian wild reindeer centre pavilion

35. Iwan Baan / Harbin Opera House

36. Nicolas Waltefaugle / Ensemble périscolaire

1. http://www.ilulissa.com/

Mise en scène

Nous venons de voir que ce qui crée une force à l'image, est à chercher plus profondément que les simples règles de composition de l'image. Un rendu n'est pas qu'une représentation d'un projet, mais une histoire. L'image doit raconter une histoire, faire imaginer des scénarios à son spectateur, afin de l'attacher à ce qu'il regarde et lui permettre de se connecter à un espace qui n'existe pas encore.

Pour lui faciliter l'attache au projet et son rendu, le contexte a une énorme force, car il structure aussi le spectateur à créer une vision du projet, et permet de l'ancrer dans une dimension actuelle et réelle. Le contexte est alors rassurant car les variables inconnues de l'image sont réduites par l'environnement connu ou reconnu par le spectateur.

L'article ***How to Create "Hotel 114" Using Cinema 4D and Photoshop*** de Ronen Bekerman sur Architizer acquiesces en ce sens. Il détaille la manière avec laquelle il insère un projet d'hotel dans un contexte urbain à Paris, proche de la rue Oberkampf (figure 37).

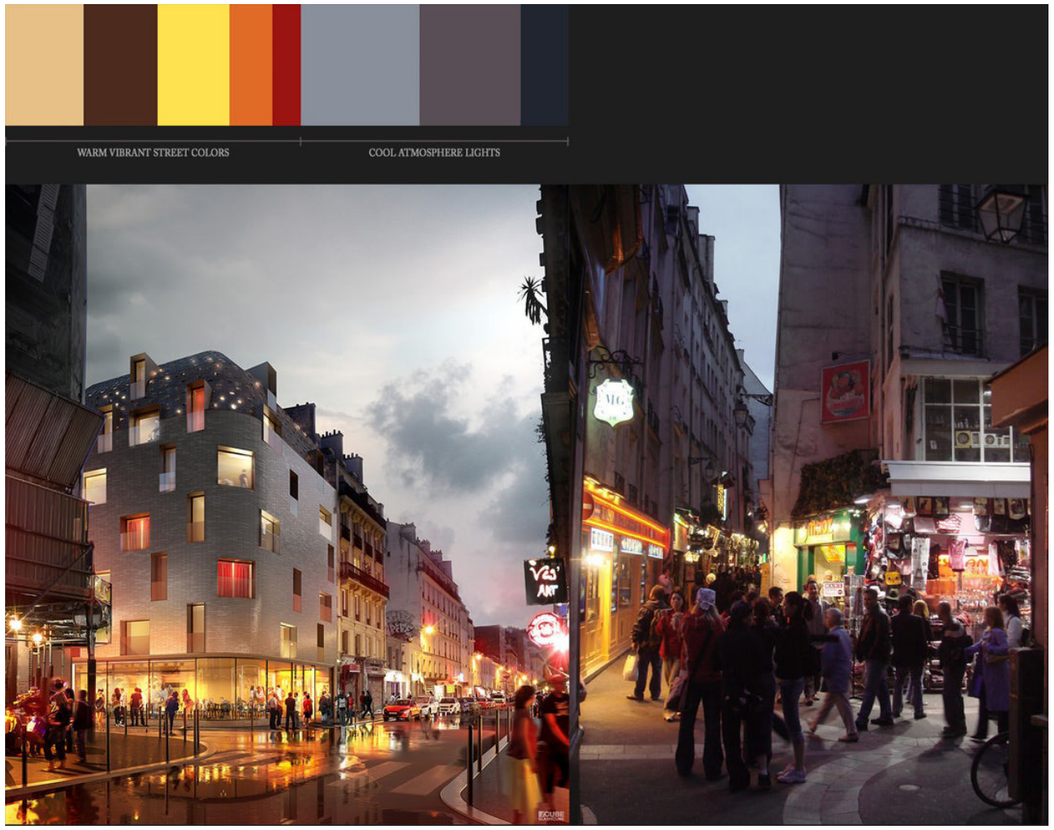
Hormis la performance et la question technique de cette illustration, il détaille aussi toute la question de la référence et de l'ambiance souhaitée de la scène, en s'inspirant de photos réelles, amenant tout un environnement et un contexte, une atmosphère et une culture du lieu (figure 38 & 39).

Cette collection d'image suggère évidemment aussi des contraintes de lumière et de matériaux, qu'il met à profit pour créer la scène. Cela permet de gérer une palette de couleurs du projet, en accord avec les intentions architecturales et l'atmosphère voulue par l'architecte.

Dès lors, le travail de l'illustrateur n'est pas que concentré sur une réalisation technique, mais aussi sur une mise en scène au service du projet [1], sur la base d'un contexte et d'une ambiance inspirée d'éléments réels, pour faciliter l'accroche du spectateur à une réalité que le projet veut se donner.

37. Rendu / Hotel 114, rue Oberkampf, Paris

38. Accumulation d'images de contextes



39. Palettes de couleurs entre rendu et références

1. The Art of rendering – How to create Hotel 114 using Cinema 4D and Photoshop / Ronen Bekerman / 29 Mars 2017

Il est plus aisé de libérer son imaginaire en présence d'une illustration portant un aspect graphique plus poussé, car la représentation peut permettre de laisser un flou d'interprétation entre le spectateur et l'illustrateur, permettant de le laisser imaginer son histoire, tout en l'encadrant.

Nous pouvons alors mettre en avant le travail de François Robin, illustrateur d'architecture spécialisé dans les dessins à main levé.

Son approche cherche à remettre en avant le travail « à la main », en se mettant en opposition au travail de l'ordinateur et la recherche de la justesse, par rapport au poète qui présente lui aussi une vérité, mais par d'autres moyens (figure 40).

La liberté offerte par le mode de représentation lui permet d'imaginer d'une manière fantasmée un projet, tout en l'insérant dans un contexte et une géométrie bien réelle [1]. Cette force graphique est au service du sens, en offrant une multitude d'histoires au sein d'une seule et même image (figure 41).

Importance du contexte

Cette question du contexte et de l'environnement est alors une donnée essentielle dans la capacité d'une image ou d'un monde à raconter une histoire. Or fabriquer une histoire et un monde est un exercice récurrent dans de nombreux médias, notamment le cinéma et le jeu vidéo.

Chacun à sa manière essaie de créer un environnement cohérent pour immerger le spectateur ou le joueur.

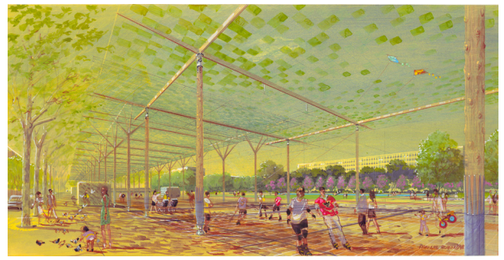
Chacun de ces médiums a sa manière de présenter son univers, qui sont toutefois parcourus par un fil rouge commun, une règle globale qui reste le principe fondateur pour chacune des décisions à prendre tout au long du processus de création de l'image, du film, du récit, de l'univers.

Joseph Kosinski est un architecte de formation aujourd'hui réalisateur, notamment de la suite du film ***TRON*** intiulée ***TRON : Legacy***.
Ce film a été suivi trois ans après par Oblivion, avec Tom Cruise.

Il présente l'architecture comme un terrain d'entraînement pour un grand nombre de champs d'exécutions différents, dont la réalisation de films.
Cela est possible par la capacité de l'architecte à penser de manière critique un projet [2], et de se donner les moyens de créer sa règle pour un projet afin de faire des choix cohérents et maintenir une cohésion du début à la fin du processus de création.

Cela fait partie à part entière de l'imagerie d'un film, la cohésion de l'ensemble des éléments dans lesquels le spectateur est plongé, que le réalisateur a créé pour donner un ton à sa réalisation (figures 42 à 45).

40. Rendu d'un parc à Confluence

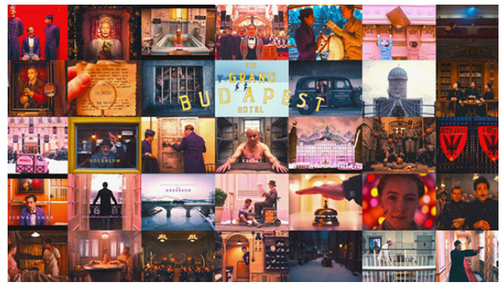


41. Halle extérieure aux Etats-Unis

42. Décor du film «Tron : Legacy»

43. Image tirée du film «Oblivion»

44. La proportion du cadre du film «Mommy»



45. La palette de couleurs du film «The Grand Budapest Hotel»

1. L'art de reprendre la main, conférence ENSANancy/ François Robin / 17 Mai 2016

2. Cutting Room: Joseph Kosinski talks to Archinect about his transition from architecture to Hollywood / Paul Petrunia / Archinect.com / Septembre 2014

La différence majeure entre le cinéma et le jeu vidéo est la manière dont le spectateur est exposé aux images. Le cinéma propose un cadre choisi par le réalisateur dans lequel il dirige le spectateur vers ce qu'il veut montrer, de la durée qu'il veut montrer.

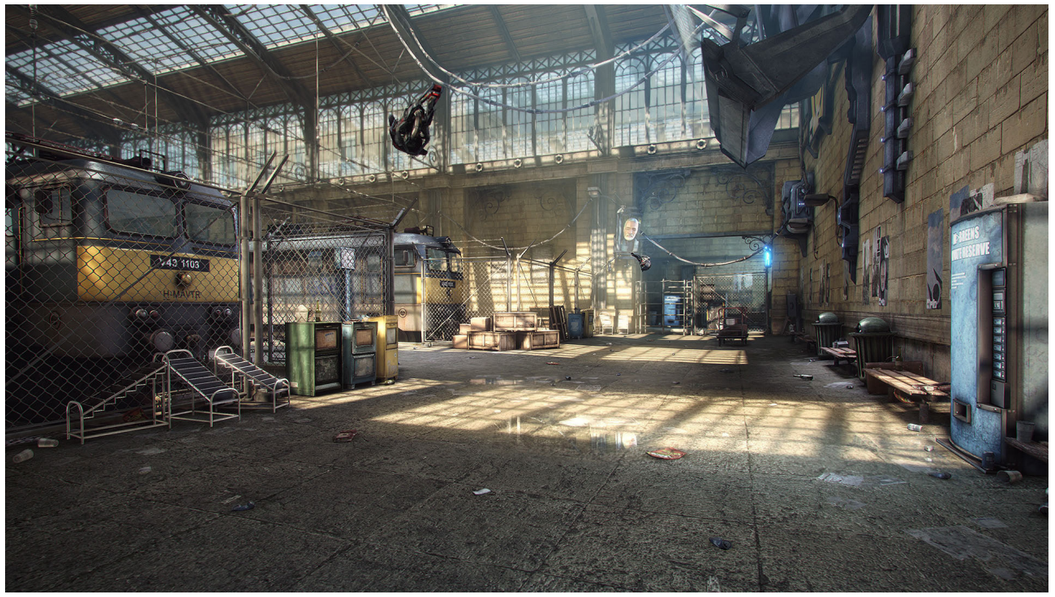
Le jeu vidéo est aussi un cadre, mais dans lequel le joueur peut se déplacer, choisir ce qu'il voit, de la durée qu'il veut. La découverte par soi-même d'un lieu et de son histoire est très important dans le jeu vidéo, car il permet d'immerger le joueur et lui donner un rôle dans l'environnement avec lequel il interagit.

De plus, le jeu vidéo a une capacité à créer de multiples environnements, de différentes manières avec différentes approches graphiques, de façon à créer des ambiances nécessaires au jeu et son histoire.

Viktor Antonov est le Game designer et compositeur du jeu ***Half-life 2*** sorti en 2004. Originaire de Bulgarie, il a puisé dans la culture de son pays d'origine pour dessiner une ville coincée entre deux cultures et deux régimes, tout comme la région des Balkans[1] (figure 46). Tous les évènements sont marqués et racontent un détail de l'histoire globale du lieu pour en devenir un outil narratif.

Cette méthode de création est inspirée du travail de Ridley Scott sur Blade Runner, film de 1982. L'environnement n'est pas une vision à un instant T, mais une suite de couches d'une histoire inventé de toutes pièces sur plusieurs centaines d'années (figure 47).

Se réinventer l'histoire sur plusieurs siècles permet de fournir des critères de design qui suivent l'idée du projet tout au long du processus de création.



46. Cité 17, Ville ou se déroule l'action du jeu «Half-Life 2»

47. Image tirée du film «Blade Runner»

1. Game Architect, émission Bits d'Arte / Rafik Djoumi /06 Janvier 2016

Arkane Studios

Arkane Studios est un studio de jeu vidéo basé à Lyon, travaillant sur la série ***Dishonored***. Le second opus est sorti le 11 Novembre 2016, toujours en collaboration avec Viktor Antonov.

La particularité de ce studio est d'engager un processus créatif intense, en utilisant des compétences de corps de métiers différents, des peintres aux sculpteurs, des architectes aux dessinateurs (figure 48).

Sébastien Mitton, directeur artistique sur le jeu, explique très clairement que le jeu est une œuvre à part entière[1], qui doit gérer non seulement la question visuelle, mais aussi le son, l'histoire, les volumes et les espaces parcourus.
Son rôle est de former une vision commune et globale du projet, et de la maintenir tout au long du processus créatif. Tout comme le projet, la question de la référence est essentielle pour se comprendre et faire réagir le joueur, en rapport avec sa propre culture.

Le jeu vidéo est alors absolument capable de créer un environnement, de se lier à l'architecte par la création d'une ambiance et d'une atmosphère d'un lieu en intégrant une histoire qui permet de dialogue avec le spectateur.
Tous ces points sont des fondamentaux du rendu, de quelque sorte qu'ils soient.

Maitriser tous les aspects artistiques est un premier pas, mais il doit être en corrélation avec la technique nécessaire pour la réaliser.

Le jeu a été réalisé sur le ***Void Engine***, une variante du moteur ***Id Tech 5***, modifié de manière à pouvoir améliorer les aspects graphiques nécéssaires à la vision du jeu par l'équipe, notamment l'interaction lumière / objet (figure 49).

L'architecte possède déjà cette culture artistique et l'approche sensible d'un projet en accord avec les principes qu'il a défini. L'aspect technique est, par contre, souvent éludé, par manque de temps, par manque de formation ou par refus de le comprendre. Hors, nous venons de voir que pousser la question de l'art en lien avec la technique est la clé de la réussite visuelle et sensible d'un rendu, d'une vision que l'on souhaite faire passer.

« Sous prétexte que ce sont des jeux vidéo, on ne pourrait pas faire appel à d'autres expressions artistiques qui viendraient justement enrichir les jeux vidéo ? »
Sébastien Mitton, Directeur Artistique sur Dishonored 2

Cette phrase peut être retourné pour l'appliquer à l'architecture, ne pourrait-on pas faire appel à des compétences et des moyens du jeu vidéo pour enrichir le projet ?

Le monde du jeu vidéo a les moyens de construire ses visions, de par une culture visuelle, mais surtout une culture technique au service d'une volonté créative (figures 50 à 53). Il existe des dizaines d'approches visuelles différentes dans les jeux vidéo, toutes au service d'un gameplay ou d'une histoire différente.

Les solutions techniques développées par le jeu vidéo pourraient servir à l'illustration d'architecture, de par le résultat graphique permettant une grande liberté et l'efficacité des procédés mis en œuvre pour arriver à ce résultat.

Nous allons donc voir comment ces solutions peuvent être transformées pour l'architecte, avec des moyens limités, autant temporel que matériel.

48. Sculpture de bustes de personnages

49. Influence de la lumière sur une scène

50. Team Fortress 2

51. Journey

52. Overcooked

53. Assassin's Creed : Unity

1. Dishonored 2 ou l'Art au service du jeu : Rencontre avec Arkane Studios / Olivier Pallaruelo / 06 Novembre 2016

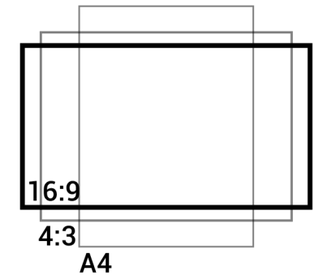
FICHE RAPPEL
ANALYSE PHOTOGRAPHIQUE

SENS

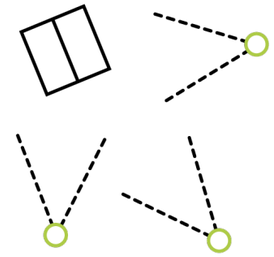
Les différents éléments fabriquant l'image servent à créer un sens et un but à la représentation.
La difficulté étant de faire passer son message correctement, afin de pas perdre la volonté première de l'image.

CADRAGE

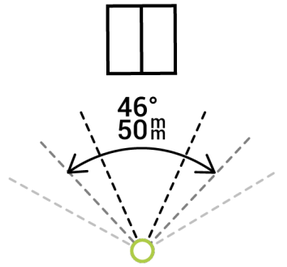
COMPOSITION



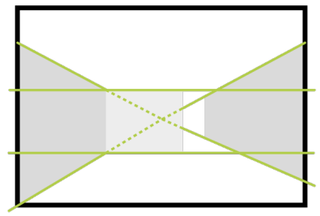
Format de l'image



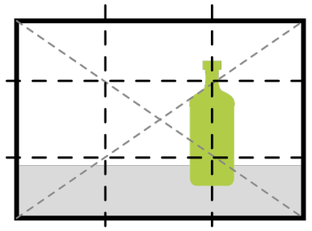
Point de vue



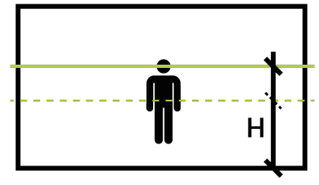
Focale



Lignes directrices



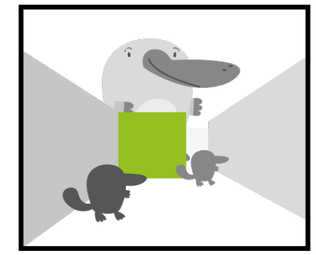
Règles de compositions



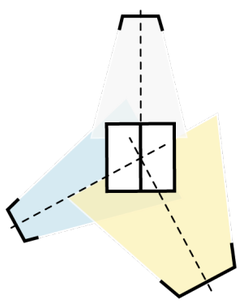
Hauteur des yeux

Plans

Echelle



Occupation du volume



Eclairage

Colorimétrie

Ambiance

PARTIE 2 :
LES OUTILS ISSUS DU JEU VIDEO

A Une histoire d'optimisation

L'avancée en terme visuel est à mettre en relation avec les optimisations technologiques des supports informatiques. La question de l'optimisation est déjà prise en considération à l'apparition des premiers jeux vidéos grands publics dans les années 80.

En septembre 1985 est disponible le jeu ***Super Mario Bros*** sur la NES (Nintendo Entertainment System), première console développée, construite et distribuée par Nintendo.
A cette époque, la cartouche contenant le jeu était d'une capacité de 256 Kilooctets, c'est à dire mille fois moins qu'un petit jeu en 2D actuel, d'une taille d'environ 250 Mégaoctets. Toutefois, certains jeux actuels peuvent atteindre une taille d'environ 70 Gigaoctets.

Dès lors, certains élément visuels 2D, ou sprites, sont utilisés plusieurs fois, avec des rôles différents. Par exemple, les buissons et les nuages sont les mêmes sprites, avec une simple variation de couleur (figure 54).

Afficher des éléments à l'écran requiert de la puissance de calcul, qui était beaucoup moins importante à l'époque qu'aujourd'hui. Pourtant, cela n'a pas empêché quelques jeux de faire parler d'eux pour leurs exploits techniques. Nous pouvons citer ***Battlezone*** en 1980, qui utilisait des éléments vectoriels pour afficher un monde 3D, uniquement composés de tracés, sans textures.

A ce moment, le fait de pouvoir expérimenter une 3D naissante était suffisant pour n'importe quel spectateur, même si les conditions d'expérimentations pourraient aujourd'hui être qualifiés d'affreuses [1], avec une fréquence d'affichage de l'ordre d'une image par seconde et un aspect visuel négligé.

Nous passions alors d'une 3D composée de fils de fers, à une 3D vectorisée, remplissant les faces par des couleurs uniques (figure 56).

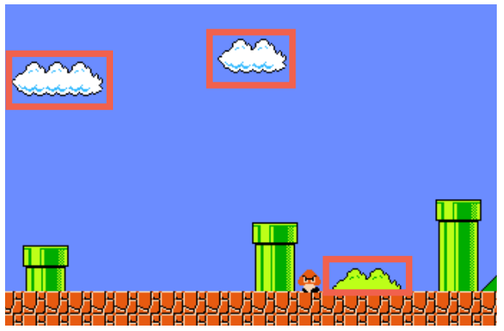
C'est l'apparition du standard VGA crée par IBM en 1987 qui va permettre une avancée dans l'affichage des données sur les écrans. En effet, le VGA (Video Graphics Array) offrait une résolution d'écran de 640x480 pixels et jusqu'à l'affichage de 256 couleurs.

C'est à partir de ce moment que nous pouvons commencer à parler de la texture tel quel nous l'utilisons aujourd'hui, c'est à dire une image travaillée appliquée sur un objet en trois dimensions.

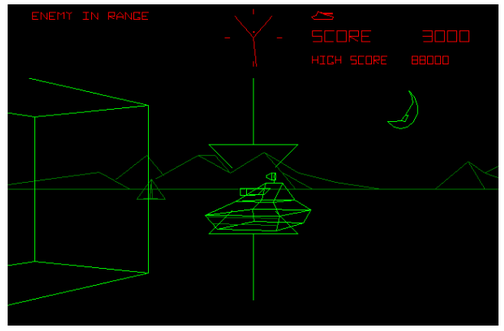
Nous retrouvons alors quelques jeux précurseurs, expérimentant les nouvelles possibilités offertes par ces avancées techniques, tels que ***Ultima Underworld*** ou ***Wolfenstein 3D***.
Toutefois, ces jeux contournaient quelques difficultés, en n'affichant qu'un environnement en trois dimensions et des ennemis animés en 2D dimensions, toujours par l'utilisation de sprites (figures 57, 58 & 59).

Les premiers jeux en 3D complètes ne sont réellement qu'apparus en 1995 avec la 5eme génération de consoles , la Playstation 1, la Nintendo 64 et les premières cartes graphiques en 1997 (figure 60).

1. PAUSE PROCESS : La texture / 24 Juin 2015 / Youtube.com



54. Super Mario Bros

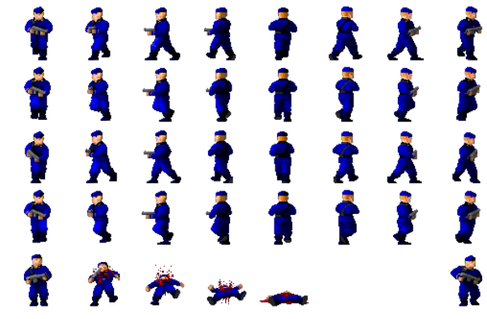


55. Battlezone

56. StarFox

57. Ultima Underworld

58. Wolfenstein 3D



59. Sprites d'animation d'un ennemi de Wolfenstein 3D

60. 5eme génération de consoles

Principes

Les objets 3D et les textures sont des éléments indissociables à notre époque. Toutefois, la modélisation requiert une attention particulière, et doit être travaillée avec la texture pour maintenir une optimisation des moyens et éviter des traitements de données trop lourds pour le support informatique.

Il existe plusieurs moyens de créer un objet 3D, la méthode la plus connue étant l'utilisation de polygones simulant une surface sur laquelle une texture sera appliquée (figure 61). C'est cette méthode qui est la plus utilisée aujourd'hui puisqu'utilisée majoritairement dans l'industrie du jeu vidéo.

Une autre méthode est l'utilisation du voxel, abréviation de ***Volumetric Pixel*** pour modéliser un objet ou un environnement. Cette méthode consiste à créer un objet par la multiplication d'un module, comme une brique sur un mur[1]. Elle a notamment été utilisée au début des années 90, lorsque la question graphique n'était pas aussi développée qu'aujourd'hui. Cela permettait de construire un environnement plus simplement par la répétition du module, plus ou moins gros, en allégeant la quantité d'éléments à l'écran.

Le polygone a très vite été popularisé par la constance recherche d'un réalisme graphique, poussant les développeurs à travailler les textures sur des modèles 3D allégés en faces, plutôt que d'augmenter le nombre de voxel d'un modèle pour arriver au même résultat.

Un objet en trois dimensions est composé de plusieurs éléments, s'assemblant pour obtenir l'objet final (figure 62).

Les ***vertices*** sont les points qui vont donner les coordonnées principales de l'objets. Les vertices sont ensuite reliés par des ***edges***, qui vont former des faces. Assemblées, les faces donneront des polygones, qui à leur tour formeront les surfaces du modèle.

Le principe de base de la modélisation repose sur le fait de ne dessiner que des polygones à trois ou quatre côtés, appelés respectivement ***Triangles*** et ***Quads***. Pour effectuer le rendu d'un objet et de sa texture, le moteur se chargera de trianguler le modèle et le nombre de polygones va affecter le temps de rendu.

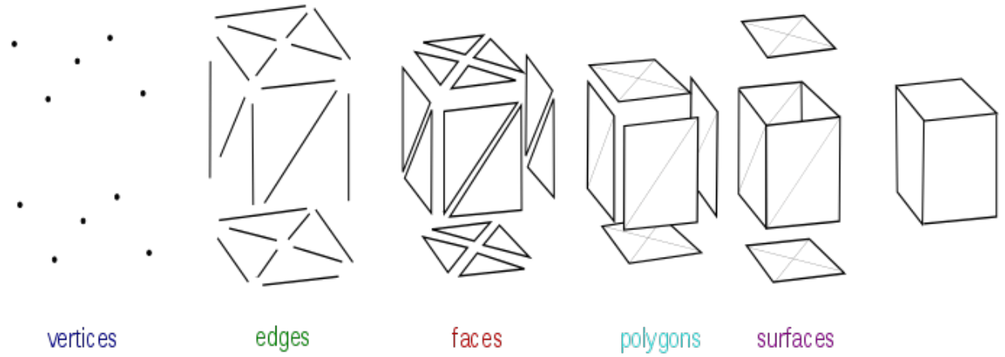
Dès lors, limiter les polygones à quatre côtés permet au moteur de n'avoir qu'une seule opération de triangulation à faire [2], en n'ayant que deux choix de triangulation (figure 63).

Alors qu'augmenter le nombre d'edges au-delà de quatre augmentera le nombre de manière de trianguler, affectant la puissance de calcul (figure 64).

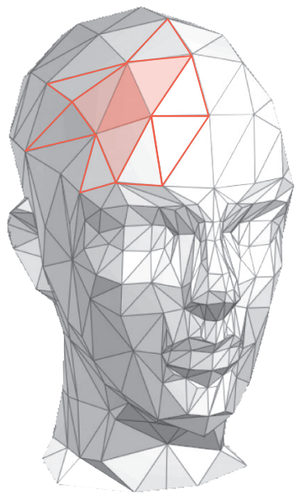
Les moteurs de jeux sont très sensibles à cet aspect, car l'objectif est de rendre un minimum de 24 images par secondes pour assurer une fluidité à l'écran, contrairement à un rendu tel que nous le concevons pour l'architecture, où une seule et unique image peut mettre plusieurs heures à être calculée.

1. PAUSE PROCESS : La modélisation / 12 Septembre 2015 / Youtube.com

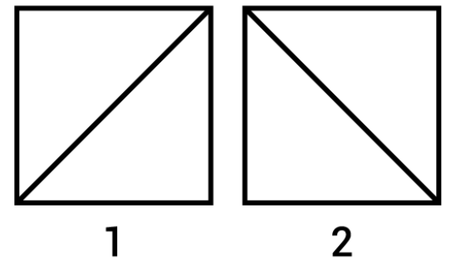
2. 3D Primer for Game Developers: An Overview of 3D Modeling in Games / Mars 2013 / https://gamedevelopment.tutsplus.com



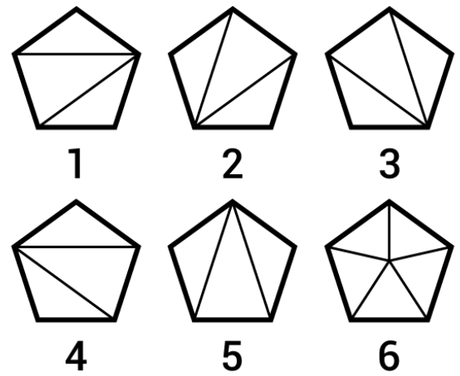
62. Elements formant un objet 3D



61. Modélisation d'une tête par polygones



63. Trianguler un «Quad»



64. Trianguler des polygones comportant plus de côtés

High Poly / Low Poly

Toutefois, les avancées en terme de performances permettent une plus grande liberté dans l'affichage des objets 3D et cela se vérifie par la comparaison de modèles similaires.

Par exemple, le modèle du personnage de Mario était composé de 752 Triangles à ses débuts dans Super Mario 64 en 1996[1], puis 4718 Triangles en 2001 pour ***Super Smash Bros Melee*** et enfin parvenir à un modèle de 10656 Tri pour ***Mario & Sonic aux Jeux Olympiques*** en 2014 (figure 65).

Cette recherche de l'optimisation du temps de rendu s'effectue de plusieurs manières, avec l'utilisation de modèles plus ou moins détaillés selon la distance, soit par une technique appelée LODs ou par la subdivision dynamique du modèle.
Les LODs, ou ***Level of Detail*** est une technique permettant de charger un modèle plus ou moins détaillé selon la distance à laquelle il se trouve de la caméra.

Cette méthode est utilisée depuis les premiers jeux en 3D, puisque nous retrouvons déjà des modèles dits ***Low Poly*** dans ***Super Mario 64*** en 1996 (figure 66).
Le nombre de triangles est alors divisé par deux pour le modèle, permettant d'alléger le système de manière plus efficace.
Dans le cas du modèle de Mario dans ***Super Mario 64***, le modèle Low Poly ne contient que 208 triangles, soit approximativement trois fois moins que le modèle complet. Dans le cas du jeu ***Super Smash Bros Melee*** de 2001, nous passons de 4718 à 332 triangles pour le modèle Low Poly, soit un multiplicateur de 14.

Optimisation avancée

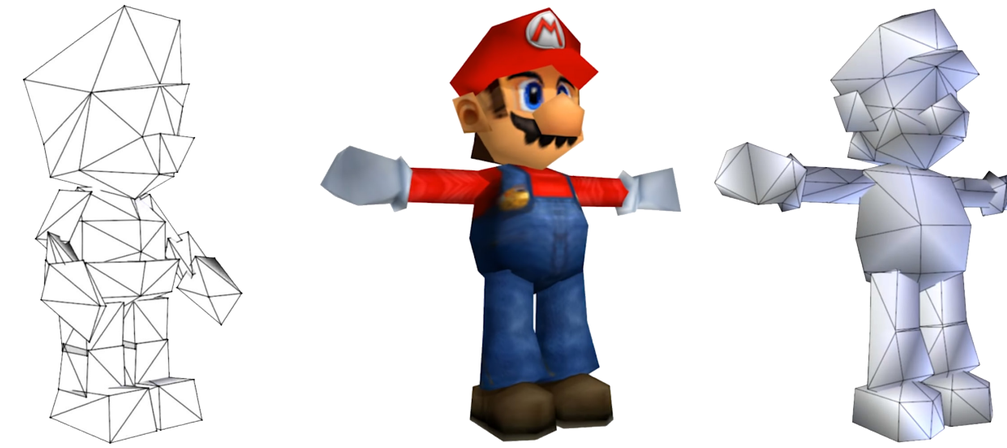
Le système de LOD fonctionne par itération du modèle de la distance, d'un modèle LOD0, le plus détaillé, à un modèle LOD2, 2, ou 3 selon la volonté du développeur [2].

L'optimisation se fait alors par le nombre de polygones, mais aussi par le shader utilisé et la texture. Par souci de simplicité, la texture et l'UV sont gardés similaires pour l'ensemble des modèles, mais cela provoque une texture trop lourde pour le modèle associé.

Certains jeux plus récents vont plus loin et utilisent même un système ayant la capacité de charger ou non les éléments visibles, comme nous pouvons le voir très récemment dans le jeu ***Horizon Zero Dawn***, sorti en Février 2017 (figure 67).

Pour préparer au mieux l'aplat de texture, il faut au préalable réfléchir son modèle, en se questionnant sur les faces à montrer ou non et si des faces seront vues et texturées, penser à l'endroit où la couture se fera pour déplier le modèle en créant un UV.

65. De gauche à droite : Super Mario 64 (1996), Super Smash Bros Melee (2001), Mario & Sonic aux jeux olympiques (2014)



66. De gauche à droite : Super Mario 64 (1996), Super Smash Bros Melee (2001) texturé et filaire

67. Chargement dynamique en fonction de la caméra

1. Super mario - Low Poly / Avril 2017 / Youtube.com
2. PAUSE PROCESS : L'optimisation / Juillet 2017 / Youtube.com

C UV Mapping

La modélisation terminée, il faut ensuite affecter une texture, une couleur au modèle 3D. Il existe plusieurs manières d'appliquer une texture sur un modèle 3D. La technique la plus connue et utilisée est l'UV Mapping.

Nous avons vu précédemment qu'un modèle est composé de plusieurs éléments, des vertices jusqu'aux faces créant le volume, chacun de ses éléments amène à un objet en trois dimensions. Le principe de l'UV Mapping est d'utiliser ses éléments comme références géométriques pour appliquer une texture sur ses faces.

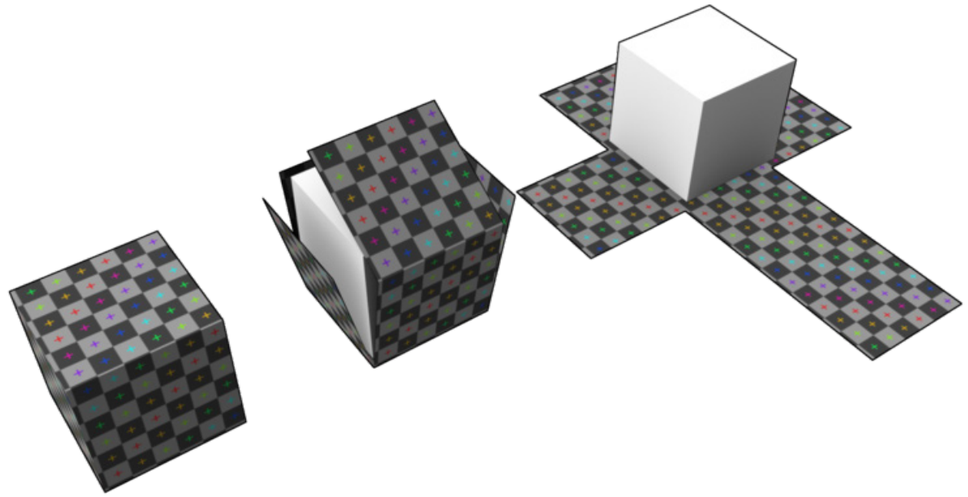
L'expression ***UV*** découle du fonctionnement en deux dimensions du processus, U et V représentent les axes du plan en deux dimensions de la texture à appliquer. Le choix de U et V s'explique simplement par le fait que les axes X,Y,Z sont déjà utilisés pour les trois axes du modèle 3D.
L'UV Mapping est très proche du principe des patrons de découpe utilisés par exemple en couture ou pour la réalisation d'éléments décoratifs en papiers.

Par exemple, dans le cas d'un simple cube, les faces dépliées peuvent donner un résultat équivalent à celui d'un patron réalisé pour fabriquer un dé en papier (figure 68).

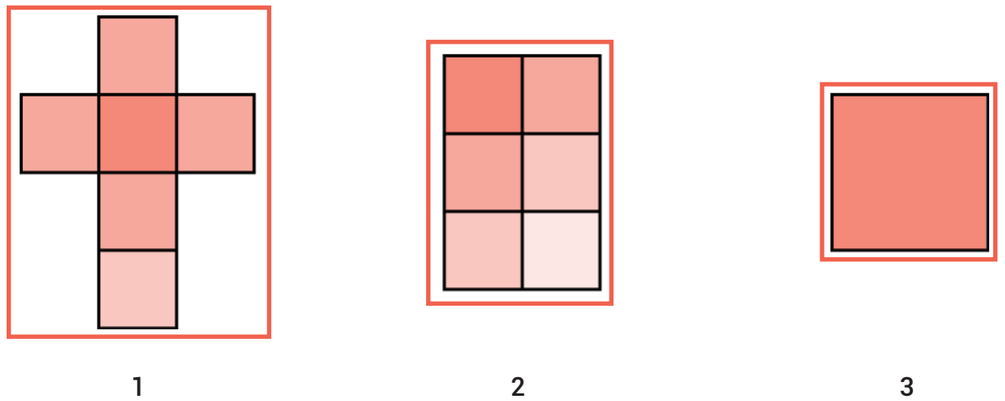
La difficulté réside dans le choix des coutures pour être efficace dans le traitement des textures. En effet, contrairement à la réalité, nous pouvons nous permettre de déplier un objet de la manière que l'on souhaite, même le cube peut être déplié de plusieurs manières (figure 69).

Le premier patron présente un modèle déplié selon une logique physique, et comprend alors beaucoup d'espace inutilisé pour la texture appliquée au modèle.
Le second patron regroupe toutes les faces afin de maximiser l'espace sur la texture et dans le même temps permettre à chaque face du cube d'avoir un traitement différent.
Le dernier patron se rapproche du second par son optimisation de la surface, mais est différent par la manière de la gérer. En effet, dans ce cas particulier, toutes les faces sont superposées, ce qui permet d'utiliser une seule et même texture pour toutes les faces, tout en réduisant sa taille.

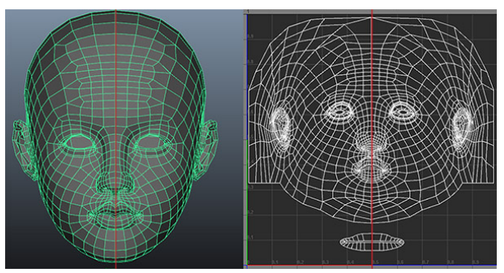
Dès lors, il s'agit d'être intelligent dans la manière de développer son patron, afin d'avoir un maillage complet sans couture lorsque les textures sont apparentes et privilégier les coutures aux endroits non vus et invisibles pour l'utilisateur (figure 70 & 71).



68. Dépliage d'une texture autour d'un cube



69. Différentes manières de «déplier» un cube



70. Dépliage d'une visage

71. Texture d'un visage déplié

Définition

Le mot texture nous vient du latin ***Textura***, désignant un objet tissé. Or, l'action de tisser crée une surface imparfaite, pleine d'aspérités, de différentes couleurs ou de motifs[1]. Et c'est le but d'une texture, simuler une matière appliquée à un objet.

Pour parvenir à un résultat convainquant, les technologies se sont développées afin de créer des systèmes gérant plusieurs textures pour un seul objet, en les combinant pour afficher des effets supplémentaires, tels que la brillance, le relief, des variations de couleurs, etc.

Toutes les textures sont regroupées dans un ***Shader***, un programme utilisé et conçu pour décrire comment une surface va être rendue, à partir de paramètres d'entrée, dont les textures[2].
Ils sont majoritairement utilisés pour des rendus en temps réel et donc les jeux vidéos, afin de permettre une efficacité de rendu et de performances, offrant une visualisation d'au moins 24 images par seconde assurant une fluidité.

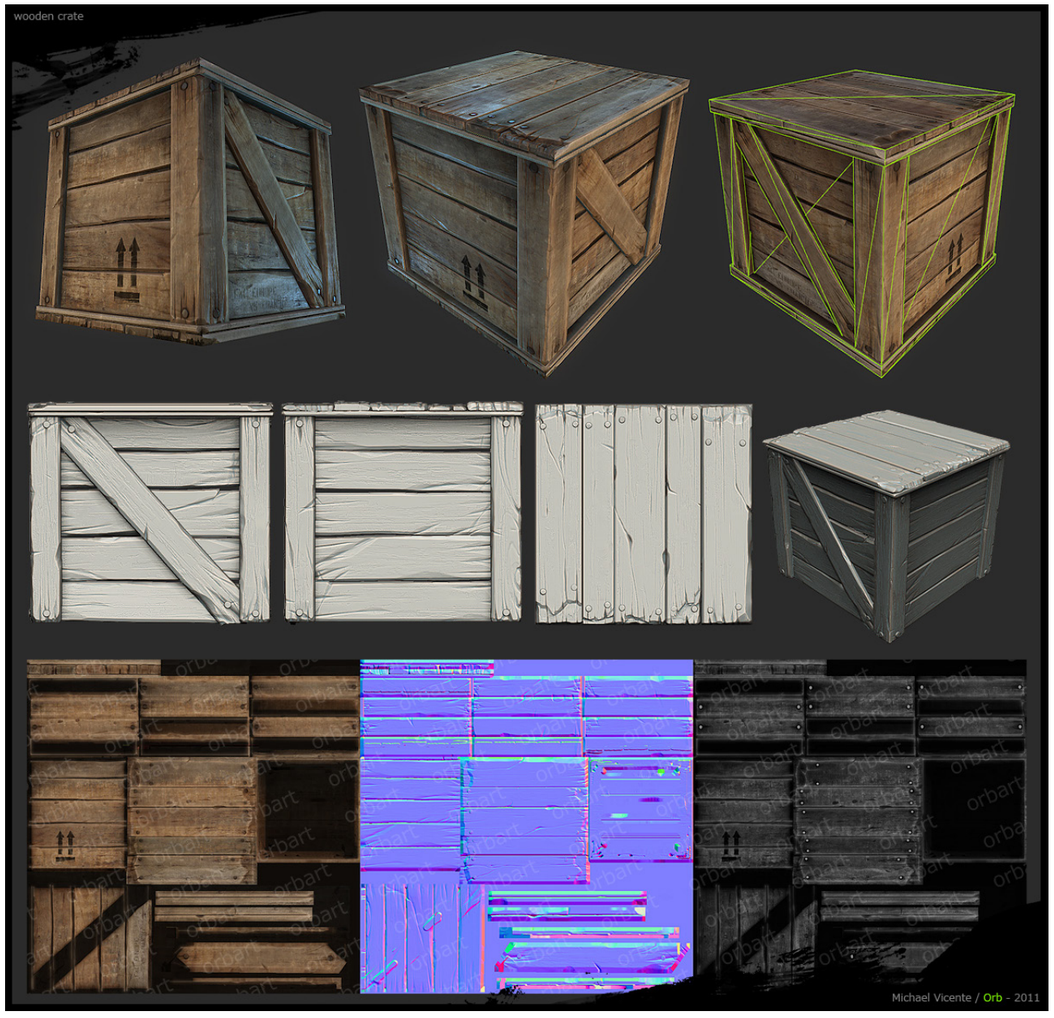
Toutefois, certains moteurs de rendus utilisent des shaders beaucoup plus précis, ne permettant pas une application au jeu vidéo, puisqu'il sera nécessaire de calculer chaque image pendant un temps beaucoup plus long.

Un matériau est alors l'ensemble de paramètres qui définissent la couleur, la brillance, la rugosité d'une surface. Il est aussi l'interface avec l'utilisateur d'un shader, qui lui se chargera de contrôler le rendu de cette même surface, selon les paramètres définis par l'utilisateur (figure 72).

La difficulté est de fournir un classement des différents types de textures existants, car chaque moteur a son propre système, ses propres besoins et sa propre déclinaison de textures. Il n'y a pas de classement précis de tous les acteurs intervenant sur les types de textures. Cependant, nous pouvons retrouver quelques familles et quelques types de textures étant le plus souvent utilisées.

Avant d'appliquer tout effet sur la surface, il faut afficher une couleur. Nous commencerons donc par les ***Color Maps***.

1. PAUSE PROCESS : La texture / 24 Juin 2015 / Youtube.com
2. Polycount - Shaders / 27 Mai 2016 / http://wiki.polycount.com/wiki/Shaders



72. Plusieurs textures appliquées sur un modèle pour gérer les effets face à la lumière

Color Maps

La ***Diffuse Map*** ou parfois appelée ***Albedo*** est la texture de couleur la plus utilisée. C'est elle qui donnera les couleurs de la surface selon l'image de la texture qui lui sera appliquée. Elle n'ajoute aucun effet au matériau (figure 73).

La ***Detail Map*** est aussi une texture de couleur, qui a pour unique objectif de rajouter du détail à une autre texture, notamment en la multipliant plusieurs fois sur une surface. Elle a l'avantage d'amplifier le contraste et des éléments sans modifier la ***Diffuse map*** et de ce fait, permet d'être gérée indépendamment par le moteur, en ne l'affichant que selon une certaine distance afin d'éviter les artefacts et réduire la consommation des ressources par exemple (figure 74).

Il existe aussi des textures appelée ***Gradient map***, dont l'unique but est de changer la teinte d'une texture par un masque de couleur.

Specular Maps

Le mot ***Specular*** est traduit en français par spéculaire, un adjectif désignant la qualité réflective d'un objet. Toutes ces textures ont comme but de donner des indications sur la manière de gérer les effets de lumière de l'objet au shader.

La ***Specular Map*** est alors la plus utilisée. Cette texture contrôle l'intensité et la couleur des reflets et effets de lumière sur l'objet. Cette texture est en nuances de gris, avec le noir ne reflétant peu ou rien et le blanc reflétant le plus de lumière (figure 75).

La ***Gloss Map*** fonctionne conjointement avec la specular map, en lui désignant l'état de la texture sous les reflets, plus ou moins brillant selon les nuances de gris de la texture. En effet, la lumière pourra se refléter de manière homogène si le matériau réfléchit la lumière de manière diffuse, mais aussi de manière plus précise s'il s'apparente à du verre.

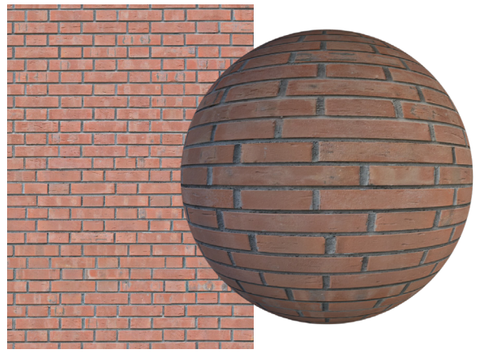
La ***Roughness Map*** englobe l'aspect global d'un objet, en donnant les informations concernant la rugosité de l'objet. De cette manière, il est possible d'avoir un objet présentant des faces rugueuses et des faces lisses, affectant alors la manière dont la lumière interagit. Cette texture est aussi composée de nuances de gris, le noir pour l'aspect rugueux, le blanc pour l'aspect lisse (figure 76).

La ***Metallic Map*** définit quelles parties du modèle sont métalliques ou non. La différence est très importante pour le rendu, car les métaux, n'ont pas les mêmes propriétés face à la lumière que la quasi-totalité des autres matériaux (figure 77).

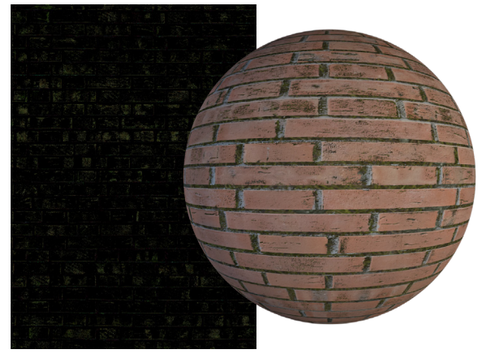
Il y a alors une différence à faire entre les matériaux dits ***Dielectic*** et les matériaux ***Metallic***. Un matériau dit ***Dielectic*** reflète la lumière de manière plus diffuse et légère qu'un métal et la diffusion de la lumière de manière plus globale nous fait ressentir sa couleur plus intensément.

Un matériau ***Metallic*** ne diffuse pas sa couleur par la lumière, mais présente toujours des reflets de couleurs propres au métal. Par exemple, un matériau cuivré sera toujours teinté rouge / orange et la couleur de la lumière n'affectera que très peu cet effet.

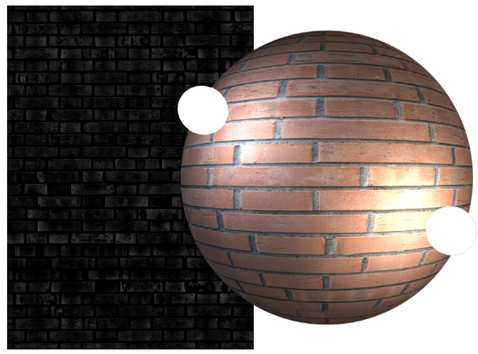
Il existe d'autres types de texture pour gérer les effets de la lumière sur un objet, comme les ***Anisotropic Maps***, utilisées pour donner une forme aux reflets, notamment sur les éléments fins comme les cheveux ou encore les « BDRF Maps », forçant une forme et une direction aux reflets, utilisés pour économiser des ressources mais surtout pour donner des aspects cartoons aux modèles.



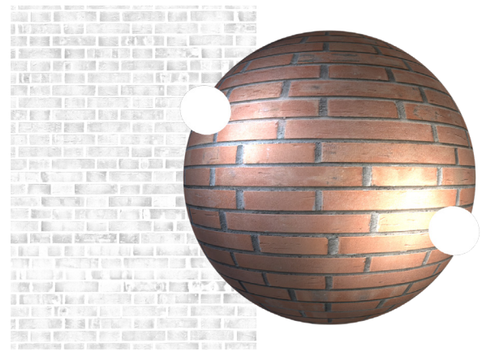
73. Diffuse Map appliquée à une sphère



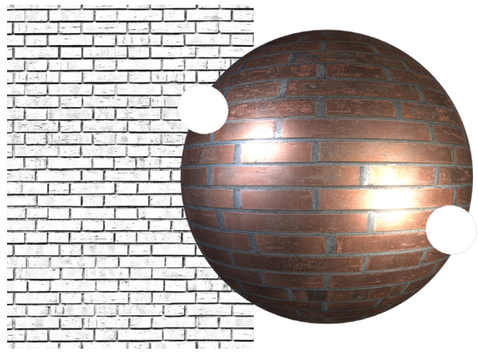
74. Detail Map combinée avec une diffuse map pour ajouter du détail



75. Specular Map appliquée à une sphère



76. Roughness Map appliquée à une sphère



77. Metallic Map appliquée à une sphère

Bump Maps

Dans cette catégorie sont replacés les types de textures ayant comme but de produire des effets de reliefs sur le modèle.

La texture la plus connue est la ***Bump Map***, une texture en nuances de gris, simulant l'effet de relief sur un objet, en trichant avec la lumière. Si elle a été remplacée, c'est notamment pour ses limitations provenant de sa texture en valeurs de gris, n'offrant plus assez d'informations pour créer un relief réaliste (figure 78).

La texture lui succédant est nommée ***NormalMap*** et reprend les principes, mais de manières plus précise et plus développée. Chaque pixel garde une couleur, donnant les informations sur la direction de chacun eux à simuler sur la surface. Son utilisation a été popularisée car elle permet de simuler beaucoup d'effets, sur un maillage très léger (figure 79).

De plus, une normal est crée à partir de deux modèle 3D, un modèle dit ***High Poly*** qui sera le modèle 3D de référence et un modèle ***Low Poly*** sur lequel sera appliquée la Normalmap, qui n'est rien d'autre que la représentation graphique de la différence entre les deux modèles.

L'augmentation des performances a permis de créer des nouveaux outils pour déformer le maillage de manière plus conséquente, au lieu de le simuler.

Les ***Displacement Map*** sont des textures en nuances de gris ayant comme objectif de déformer une surface et d'en faire un relief 3D véritable. Toutefois, si elle n'est pas utilisée avec des techniques de subdivisions dynamiques, le modèle peut tout de même devenir très lourd (figure 80).

Avant les nouveaux outils de Displacement, certains jeux utilisaient des ***Height Maps*** gérant les hauteurs des différents points d'une surface pour créer un terrain. Les paysages virtuels utilisent souvent une ***height map*** dessinée en 2D pour créer un environnement.

Il existe d'autres types de textures pour gérer le relief, mais elles sont le plus souvent associées à un moteur en particulier ou une technologie propriétaire, empêchant ainsi la diffusion de l'outil.

Nous pouvons citer les ***Parallax Maps*** crée avec l'arrivée des nouvelles librairies DirectX 11, les ***Vector Displacement Maps***, ***DuDv Maps***, ***Flow Maps*** ou encore ***Curvature Maps***.

Light Maps

Les specular maps ont comme but de gérer les effets de la lumière sur un objet, les light maps ont comme objectif de produire et contrôler la lumière depuis un objet.

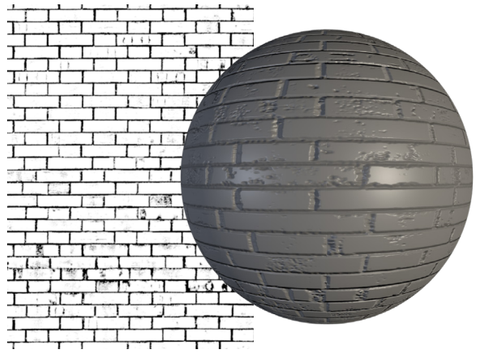
La plus simple étant l'***Emission Map***, une texture montrant les éléments lumineux d'un objet, permettant aussi d'en gérer l'intensité et la couleur (figure 81).

Une autre texture utilisée est l'***Ambient Occlusion Map***. Cette texture rajoute un effet d'ombre léger à l'objet pour asseoir sa position dans le décor (figure 82).

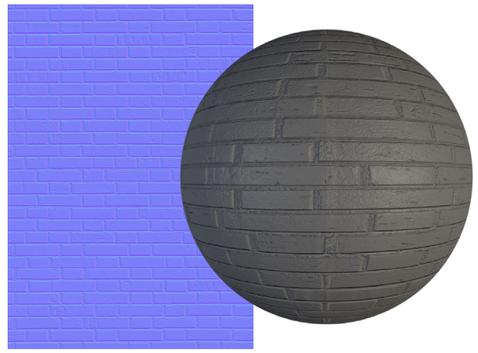
Les moteurs de jeux gèrent cet effet dynamiquement maintenant, mais cela peut tout de même être impactant en ressources, et c'est pour cette raison que cette texture existe encore aujourd'hui.

Pour des question d'utilisation de ressources, certains jeux utilisent des textures dites ***Light Map*** pour certains environnements. Le principe est de pré-calculer des éclairages d'une zone dont on sait que rien ne bougera pour les rappliquer ensuite aux décors.

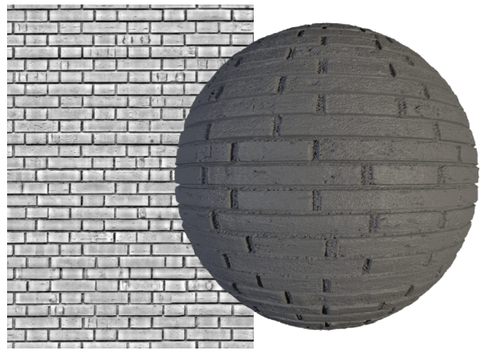
Cette méthode est alors très efficace dans des environnements fermés, où la lumière est contrôlée et présente de nombreux avantages en terme de ressources pour un jeu vidéo.



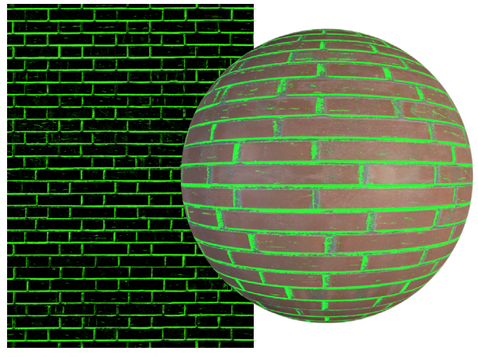
78. Bump Map appliquée à une sphère



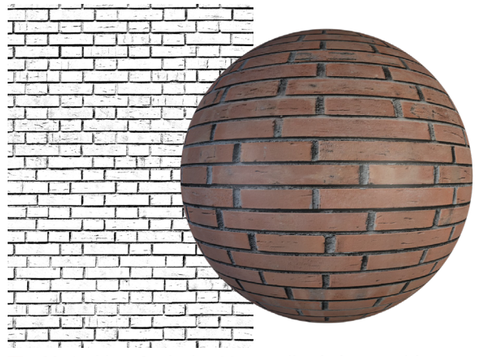
79. Normal Map appliquée à une sphère



80. Displacement Map appliquée à une sphère



81. Emission Map appliquée à une sphère



82. Ambient Occlusion Map appliquée à une sphère

Dans le cadre d'un projet architectural, il est difficile d'utiliser cette méthode, car la recherche du détail nous pousse souvent à modifier des éléments jusqu'au dernier moment, empêchant l'utilisation de tels procédés. De plus, le baking d'une telle texture prend aussi beaucoup de temps (figure 83).

Enfin, nous terminerons par quelques types de textures spécifiques.

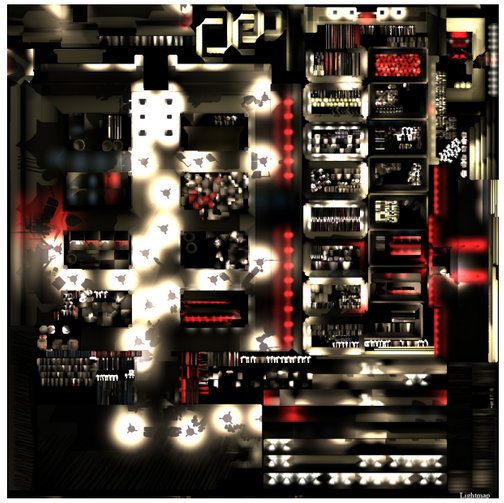
Textures spécifiques

La *Transparency Map* ou parfois appelée *Opacity Mask*, est une texture définissant la transparence ou non sur une texture. Elle est toutefois peu utilisée, sauf dans une volonté de grande précision sur des objets fins, les feuilles d'un arbre par exemple.

Le moteur est obligé de créer la transparence en analysant la texture de masque, pixel par pixel, feuille par feuille, ce qui va nécessiter beaucoup de ressources. Il est très souvent plus efficace de modéliser une feuille avec très peu de polygones, qui ne se verra que très peu dans la densité de la végétation (figure 84).

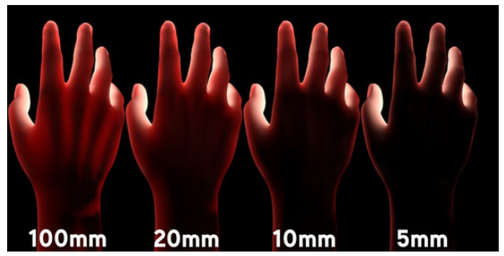
Les *Subsurface Scattering Maps*, utilisées pour les textures de peau, ou tous matériaux autorisant la lumière à légèrement passer au travers selon l'épaisseur de l'élément. L'exemple le plus simple est une main éclairée par une lampe torche. La lumière passe au travers des interstices, mais produit aussi un effet rouge le long des doigts de la main.

Cette texture gère les rebonds de la lumière à l'intérieur d'une surface semi-translucente (figure 85).



83. LightMap - Shuriken 2

84. Transparency Map



85. Subsurface Scattering Map

Introduction

Le PBR ou Physically based Rendering, est une nouvelle approche du travail de la texture et de la lumière pour les rendus.

Le changement majeur derrière cette méthode est le traitement de la lumière dans une scène. Jusqu'alors, la lumière était un défi constant pour tout les développeurs et créateurs car il n'existait pas de solutions précises pour créer un objet et l'adapter à un éclairage précis.

Nous avons vu que les différents types de textures sont apparus en même temps que les premières carte graphiques, au début des années 90.
A cette époque, les développeurs « peignaient » les effets de lumières directement sur les objets et ajustaient leurs textures ***specular*** de manière à donner une uniformité des couleurs et des reflets dans une scène.

Le PBR pose comme postulat qu'un objet ne peut réémettre plus de lumière qu'il n'en reçoit, sauf exceptions[1].
Couplé aux évolutions en terme en matière de calcul de la lumière, cela permet de créer tout les éléments d'une scène indépendamment de l'éclairage, réduisant considérablement le temps de travail puisque les objets n'ont plus d'usages singuliers, mais aussi la charge en terme de performance, puisque quelques textures vont suffire à générer un matériaux convainquant.
Il n'est alors plus nécessaire de rajouter des masques de teintes, de faux éclairages, etc.

Le PBR est une démarche de construction de la texture qui s'est développée depuis une dizaine d'années. Cette technique est un ensemble d'outils simulant la lumière de manière réaliste, un système complet de création de contenu et de rendu, pouvant accueillir des variantes selon les besoins.

Certains « ingrédients » du PBR peuvent d'ailleurs déjà être aperçus dans des productions du jeux vidéos plus anciennes, avec ***Call of Duty : Black Ops*** en 2010 et le traitement des surfaces métalliques[2] et boueuses, ou ***Super Mario Galaxy*** et le traitement du Fresnel en 2008.

Règles du PBR

Le PBR n'est toujours qu'une approximation de la réalité, mais pour s'en rapprocher, il se base sur trois points essentiels :

- La surface d'un modèle est basée sur le principe de micro-facettes
- Le matériau conserve son énergie
- Le matériau doit utiliser une fonction BRDF basé sur la physique

Nous allons revenir en particulier sur chacun de ces points.

Tout d'abord, la théorie des micro-facettes décrit une surface comme un ensemble de minuscules facettes qui reflètent parfaitement la lumière, mais dont l'alignement va dépendre du paramètre de ***roughness*** donné à cette surface.

Plus une surface est rugueuse (roughness), pour les micro-facettes vont être désordonnées sur la surface et provoquer un renvoi de la lumière dans toutes les directions de manière diffuse (figure 86).

Au contraire, une surface lisse ne sera alors pas composée d'imperfections et renverra la lumière d'une manière précise, affichant des reflets plus petits et plus précis.

Ensuite, l'approximation des micro-facettes utilise une forme de conservation d'énergie. L'énergie émise ne doit jamais dépasser l'énergie reçue, ou « un objet ne peut réémettre plus de lumière qu'il n'en reçoit ».

Plus la surface est rugueuse, plus elle va

86. Schémas représentant la théorie des micro-facettes

absorber de l'énergie et devenir sombre, par le biais des micro-facettes dispersant la lumière.

La réalité physique de la lumière nous obligerait à prendre en compte tous les rebonds de la lumière après qu'elle ait heurté une première surface, mais au prix des performances. Cet aspect physique est simplifié et la lumière est majoritairement absorbé et n'est que renvoyé sur une petite zone proche de l'impact de la lumière.

Certains shaders, déjà évoqués auparavant, prennent cette réalité physique en compte, en prenant en compte ce qu'on appelle le ***subsurface scattering***, qui va grandement améliorer la qualité sur les matériaux de peau ou le marbre, mais au prix des performances.

Enfin, la fonction dite BDRF ***Bi-Directionnal Reflectance Distribution Function*** est essentielle au PBR pour fonctionner, puisque cette fonction va définir comment la lumière est réfléchie sur une surface, en s'approchant de la réalité physique.

Cette fonction prend en compte la direction de lumière sur la surface, sa direction réfléchie, sa normale et le paramètre de rugosité pour simuler l'impact des rayons de lumière sur un matériau.

Le travail mathématique derrière ces fonctions a été établi par les studios d'animations Disney en 2007, avec un article ***Physically Based Shading at Disney***, mettant en avant le travail théorique entrepris par le studio pour créer leurs récents films, notamment ***Les mondes de Ralph***, sorti en décembre 2012[3].

Pour parvenir au résultat, il nous faut encore les textures comme paramètres d'entrée de notre fonction. Chaque moteur a sa propre manière d'appréhender le PBR, mais les textures les plus communes sont l'albedo / Diffuse, la normal, la Metallic et Roughness ainsi que l'AO Map.

Le PBR a été popularisé par Epic Games et l'Unreal Engine récemment devenu gratuit pour des utilisations personnelles. Cette méthode s'étend à de plus en plus de studios et de moteurs, faisant évoluer l'industrie du jeu vidéo et ses outils.

Blender, que nous allons voir plus tard, est en train de faire évoluer son moteur en temps réel vers le PBR.

Nous allons maintenant succinctement nous intéresser à un logiciel professionnel destiné au monde du jeu vidéo pour la création de texture, avant de faire évoluer l'approche sur Blender.

1. PBR Theory / LearnopenGL.com / https://learnopengl.com/#!PBR/Theory

2. Le PBR, la révolution silencieuse mais bien visible / Gamekult.com / https://www.gamekult.com/actualite/le-pbr-la-revolution-silencieuse-mais-bien-visible-142125.html

3. Physically-Based Shading at Disney / Brent Burley / 31 Août 2012

Substance Painter

Allegorithmic est une entreprise française basé à Clermont-Ferrand, développant des outils dédiés aux jeux vidéos. Utilisant les nouvelles technologies de rendus offertes par le PBR, elle a créé une suite d'outils pour aider la création de texture autour de modèles 3D (figure 87).

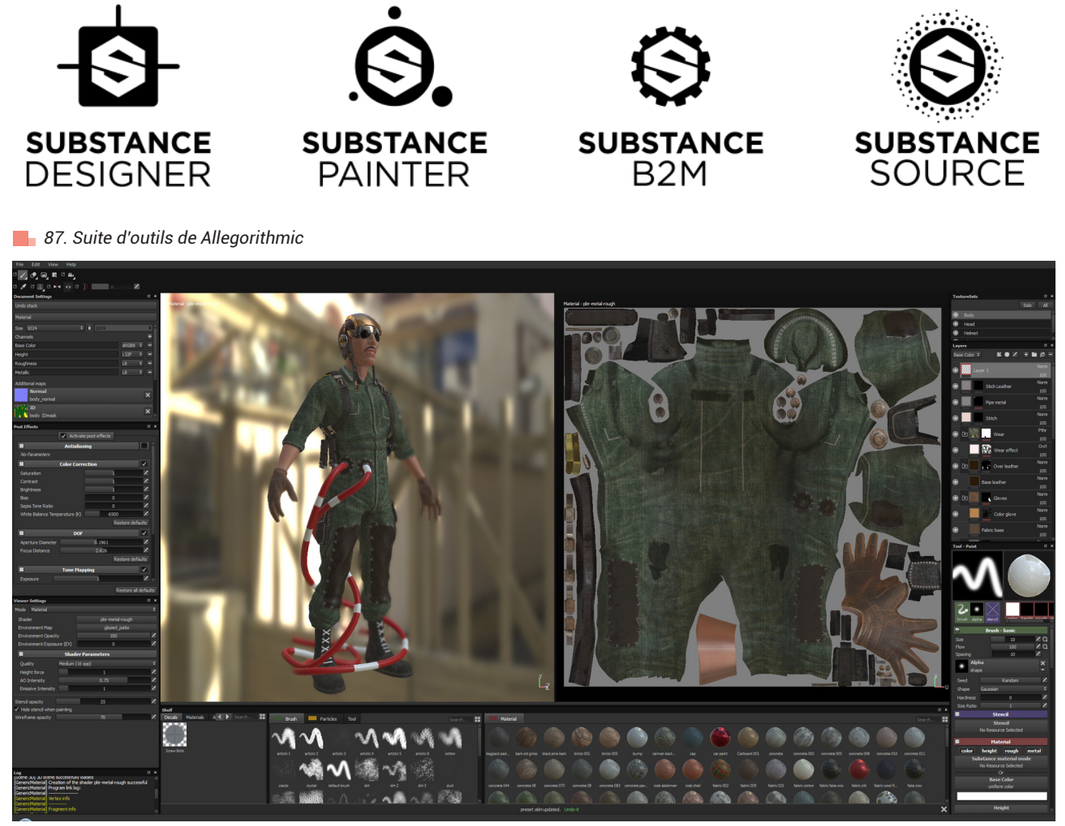
L'intérêt majeur est de pouvoir visualiser en direct l'évolution de la texture sur le modèle, et le mesurer selon différents angles, éclairages et environnements (figure 88).

L'outil de référence que je présenterai est Substance Painter, dont le fonctionnement se rapproche de Photoshop, ou d'un autre outil de traitement d'image.

La force de Substance Painter est sa simplicité d'utilisation, puisqu'il fonctionne par un système de pinceaux, textures et aplats, similaires à un outil de dessin classique, tout en offrant une multitude d'options supplémentaires par la gestion des calques du modèle, des textures paramétriques ou encore l'aplat d'effets, tels que des ruissellements d'eau directement simulés sur le modèle (figure 89).

Dès lors, la prise en compte du PBR au travers des différents types de textures qui nous avons évoqués auparavant, permet de « peindre » la texture et d'appliquer ses attributs sur tous les types de textures à la fois, diffuse, specular, normal, etc.
D'autant plus que la gestion des calques et de certaines textures paramétriques permettent de modifier simplement la texture à posteriori, afin d'en corriger des erreurs.

Cette suite dédiée aux professionnels présente d'autre outils, permettant de créer sa propre texture paramétrique ou de générer un ensemble de texture à partir d'une image.
En tant qu'outil dédié aux professionnels du jeu vidéo, nous allons chercher à nous approcher de ces méthodes et moyens par l'intermédiaire de Blender, outil de modélisation, rendu et animation, pour créer une méthode d'utilisation permettant la retouche de texture, la personnalisation d'une scène selon un flux de travail basé sur le PBR[1].



87. Suite d'outils de Allegorithmic

88. Illustration du logiciel Substance Painter

89. Bibliothèque d'éléments

1 Allegorithmic, la pépite clermontoise qui équipe les meilleurs jeux vidéo du monde, Le Monde, 29 Novembre 2016

Introduction

Blender est un logiciel gratuit et open source, incluant un ensemble de fonctions dédiées à la création 3D, la modélisation, l'animation, la simulation, le rendu, la retouche et le calibrage d'image, le montage vidéo et même la création de jeux vidéos.

En tant qu'outil open source, il existe un nombre non négligeable d'***addons***, contenu, ressources, gratuites et payantes accessibles très facilement sur internet. Nous nous contenterons des fonctions de base du logiciel et de fonctions annexes gratuites dans certains cas.

La majeure difficulté d'utilisation de Blender par rapport à un logiciel professionnel tel que Substance Painter et la gestion des différentes couches de textures en simultané. En effet, Blender ne prend pas en charge un tel système. Les textures peuvent être retouchées, redimensionnées et traitées mais de manière indépendante, ce qui rend difficile la réalisation complète d'un modèle et de ses textures à destination d'un rendu PBR.

L'objectif est alors d'utiliser des textures déjà existantes, présentant les différents types nécessaires à une utilisation dans un ***workflow*** PBR et utiliser Blender pour permettre une modification et une appropriation des textures à chaque cas particulier du projet.

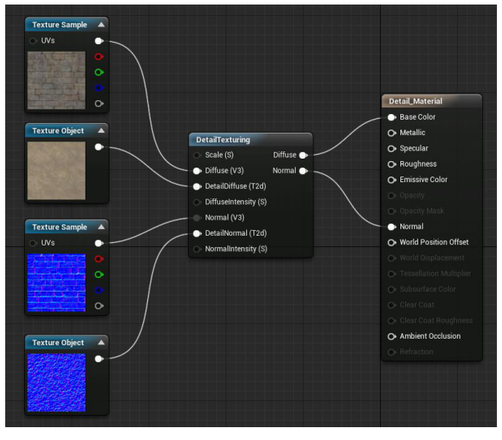
Système de noeuds

Les travaux concernant le PBR ont été intégrés à la dernière version de Blender, la version 2.79. Suivant les travaux réalisés par Disney, les équipes de développement ont créé un nouveau shader dit ***uber***, c'est à dire qu'il regroupe un assez grand nombre de fonctions permettant différentes utilisations.
Blender dispose de nombreux autres shaders, eux aussi capable d'utiliser des textures, mais nous nous concentrerons sur celui-ci en particulier, pour sa capacité à gérer les types de textures mis en avant auparavant.

Outre les shaders, Blender gère aussi plusieurs moteurs de rendus, mais nous nous intéresserons uniquement à ceux capables de gérer ces nouveaux shaders, c'est à dire ***Cycles*** et ***Eevee***, le nouveau moteur de rendu en temps réel.

Chaque logiciel de rendu, de visualisation en temps réel a sa propre interface graphique. Blender gère les matériaux selon un système de ***Nodes***, des nœuds, dont les ficelles relient les différents éléments interagissant pour créer le matériau.

Un système similaire est présent pour le moteur de jeu Unreal Engine 4, mais reste très différent de celui d'Unity 5, très classique (figure 90).
Le node, ou nœud qui nous intéresse se nomme ***Principled BDSF***. Ce shader combine plusieurs couches d'informations ou textures dans un seul et unique node (figure 91).



90. Exemple du système de nodes de Unreal Engine 4

Son avantage est de permettre une compatibilité des textures, puisque le système est similaire aux autres moteurs de rendus actuels utilisant le PBR. Dès lors, le travail effectué sur Blender peut être exporté sur Unity 5, Unreal Engine, le moteur de rendu de Pixar, Renderman et avec des textures directement exportés depuis Substance Painter, que nous avons vu un peu plus tôt.

Ce shader inclus plusieurs options et couches permettant de créer une variété de matériaux, en utilisant les types de textures présentées auparavant.

Dans notre approche de la texture, seuls quelques entrées nous intéressent.

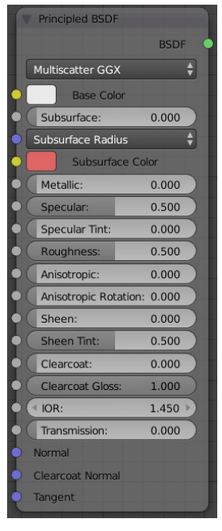
Tout d'abord, ***Base Color***, dans lequel est branchée la diffuse ou albedo maps. Cette entrée gère la couleur de la texture.
Ensuite, l'entrée ***metallic***, pour la différence entre une surface dite ***dielectic*** et une surface ***metallic***, qui ne présentent pas les mêmes propriétés face à la lumière.

Puis, l'entrée Specular, qui sera connectée à la texture ***Specular map***, en charge de définir comment la lumière se transmet sur une surface.

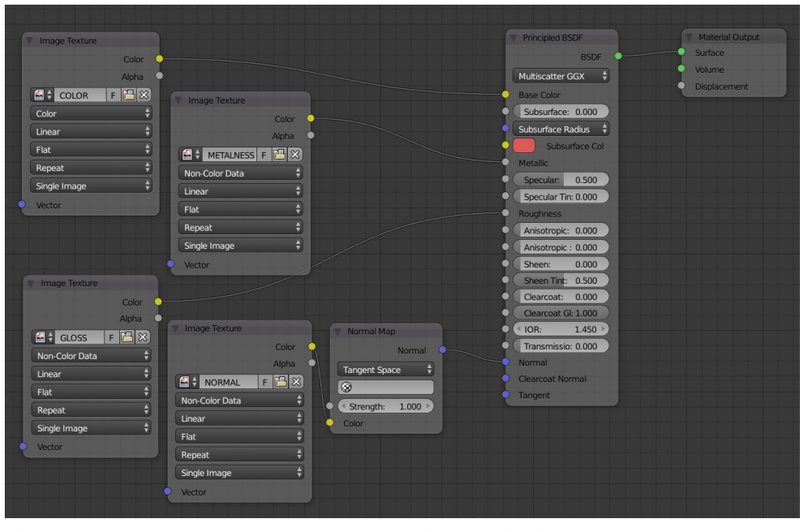
La ***Roughness Map*** sera connectée à l'entrée Roughness, en charge de la rugosité d'une surface.

Enfin, l'entrée ***Normal*** viendra se connecter à une ***Normal Map***, par le biais d'un node vecteur en charge de simuler le relief sur une surface (figure 92).

La difficulté avec Blender sera de gérer son système de nodes afin de créer le matériau que nous souhaitons obtenir. En effet, certaines textures n'ont aucun effet avec de node et dans ce cas, il sera nécessaire d'utiliser plusieurs shaders pour arriver au résultat voulu.



91. Principled BDSF



92. Schémas connexions textures sous Blender - Principled BDSF

Créer une texture sous Blender

Créer une texture est un processus créatif complexe, qui peut se faire de multiples façons, avec de multiples outils. Dans notre objectif de contenir ce travail dans un ***workflow*** basé sur Blender, nous n'aborderons ici que les outils prévus à cet effet dans le logiciel Blender.

Nous pouvons, de manière simplifiée, considérer deux approches possibles pour la réalisation d'une texture, la première, permettant de créer sa propre texture de manière autonome, grâce à l'outil ***Texture Painting*** de Blender et la seconde à partir de textures pré-travaillées compatibles PBR.

Évidemment, la maîtrise du logiciel pourra permettre un assemblage des approches, et une gestion plus fine et personnalisée du travail de texturing.

UV UNWRAPPING

La première méthode, l'outil ***Texture Painting*** nécessite de déployer l'UV du modèle, afin de pouvoir lui appliquer une texture, que nous pourrons retravailler grâce à différents outils intégrés à Blender. L'UV dépliée est visible dans la fenêtre ***UV Image Editor***, fenêtre de gestion et d'aperçu des UV, si déployées, mais aussi des textures utilisées dans un fichier (🎥Vidéo 1 00:10).

Il s'agira ensuite de déployer ou d'***unwrap*** l'UV de son modèle. Il existe plusieurs manière de réaliser cette étape, et tout dépends du modèle et des contraintes fixés par la scène. Une fois l'UV du modèle déployé, nous pouvons commencer à gérer la question de la texture (🎥Vidéo 1 00:15).

L'étape suivante consiste à créer une texture depuis cette fenêtre, en cliquant sur le bouton ***New***, puis en remplissant les champs de dimensions de la texture, le fond et surtout son nom, qui va nous permettre de l'identifier plus facilement (🎥Vidéo 1 00:35).

La dimension de la texture a une influence sur le rendu, puisqu'une texture plus lourde et par conséquence plus détaillée nécessitera un calcul plus long. De plus, l'idéal est de pouvoir organiser son UV de manière à optimiser la place prise, afin de laisser le moins de place libre possible sur la texture (🎥Vidéo 1 00:20).

La texture crée, il faut absolument la sauvegarder, en cliquant sur ***Image – Save as Image***, afin de commencer à texturer son modèle (🎥Vidéo 1 01:10).

Avant de commencer à peindre sa texture, il faut s'assurer que l'objet que nous voulons peindre a son propre matériau, utilisant le système de nodes, et un node dit ***Image Texture*** que l'on trouve dans la catégorie ***Texture***, pointant vers la texture que l'on vient de créer (🎥Vidéo 1 01:30).

Pour cela, nous allons préparer le matériau en lui donnant les instructions nécessaires afin que cette texture soit utilisée par l'objet en créant plusieurs nodes.
Le premier, ***Texture Coordinate*** a comme rôle de définir de quelle manière la texture sera appliquée, générée par Blender, par la caméra, selon l'objet, ou, dans notre cas, par les coordonnées de l'UV précédemment déployé (🎥Vidéo 1 02:10).

Le second node, évoqué auparavant, est ***Image Texture***, et sera notre rappel à la texture précédemment crée. Il faudra la sélectionner dans la liste déroulante après l'avoir connecté au premier node ***Texture Coordinate*** (🎥 Vidéo 1 02:00).

Enfin, nous allons relier ce node au shader, chargé de gérer les textures et leurs effets sur le modèle. Blender étant un outil libre, il est possible de trouver plusieurs shaders, officiels ou non, pour des usages variés.
Dans notre cas, nous utiliserons le ***Principled BDSF shader***, implanté récemment et gérant le processus PBR (🎥Vidéo 1 01:50).

Pour vérifier que la texture est bien appliquée au modèle, nous pouvons passer en ***render view***, avec le raccourci « Shift+Z » (🎥 Vidéo 1 02:18).

Texture Painting

L'outil ***Texture Painting*** intégré à Blender fonctionne de manière similaire à Substance Painter, en utilisant un système de ***brushes*** avec plusieurs effets pour appliquer des couleurs ou des textures sur une surface.

Pour ouvrir cet outil, il faut le sélectionner dans le menu déroulant ***Object Mode*** et sélectionner ***Texture Paint***. Le panneau gérant toutes les options est normalement situé à gauche de la fenêtre de prévisualisation 3D, ou accessible via la touche ***T*** (🎥 Vidéo 1 03:00).

Il faut tout d'abord vérifier que la texture de notre modèle est bien prise en compte. Si elle s'affiche sur l'objet, cela devrait être le cas, sinon, vérifiez l'onglet ***Slots***, et que la bonne texture soit sélectionnée (🎥Vidéo 1 03:07).

L'outil est décomposé en plusieurs sections, ***Brush***, ***Texture*** et ***Texture Mask***, sont les trois grandes variables pour l'aplat de couleurs ou textures sur une surface.

Brush

La fonction ***Brush*** permet de peindre sur la texture. Plusieurs paramètres sont disponibles, permettant de changer le diamètre de l'aplat ou son opacité, respectivement ***Radius*** et ***Stength***. Bien entendu, la couleur peut être changée depuis la roue chromatique, mais aussi le mode d'application de la couleur, que l'on peut considérer comme des équivalents au mode d'application d'un calque sur Photoshop. Ils existent aussi dans le ***Node Editor*** d'un matériau, permettront aussi de gérer de léger traitements d'images directement depuis Blender (🎥Vidéo 1 03:10).

La manière d'appliquer l'aplat peut se changer depuis le menu ***Stroke***, en utilisant le menu déroulant avec différentes options, dont le tracé de ligne, dégradé circulaire, etc.

L'outil Brush permet aussi différents effets pour un aplat, qui sont disponible en cliquant sur l'icone supérieure ***TexDraw***. Ces effets vont du tampon de duplication, à l'effet de flou et au remplissage d'une surface par une couleur.

Les options sélectionnées, nous pouvons commencer à peindre sur l'objet, directement depuis la visualisation 3D, et voir les aplats effectués se reporter sur la texture. Pour que cela s'applique au rendu, il ne faut pas oublier de sauvegarder sa texture.

Textures

La fonction ***Texture*** permet de charger une image afin de l'appliquer sur la texture selon les paramètres définis dans le panneau ***brush***.

Pour cela, il faut charger une texture en en créant une nouvelle par le bouton ***new***. Cette texture n'a pas encore d'image chargée, pour se faire, il est nécessaire d'aller en récupérer une depuis un autre menu, dans les onglets de l'outliner (🎥Vidéo 1 03:38).

L'onglet ***Texture*** sélectionnez, vérifiez que la texture que vous avez crée a bien le même nom que celle sélectionnée, et cliquez sur ***open*** pour aller chercher une texture de référence à appliquer sur le modèle (🎥Vidéo 1 04:38).

Une fois effectué, nous pouvons commencer à appliquer la texture sur le modèle. Le mode d'application de la texture est souvent mal choisi, et nécessité d'être changé pour plus de facilité et de compréhension (🎥Vidéo 1 05:10).

Toujours dans le panneau ***Texture***, sous ***Brush Mapping*** , un menu déroulant présente plusieurs mode d'aplat. Nous allons choisir le mode ***Stencil***, affichant la texture à l'écran, permettant de viser directement depuis la vue 3D comment appliquer la texture.

Texture Mask

La fonction ***Texture mask*** permet d'utiliser un masque à l'aide d'une texture usuellement en noir et blanc, afin d'appliquer une texture ou une couleur au travers de ce masque. Ces textures sont généralement appelées ***Alphas*** (🎥 Vidéo 1 06:25).

Nous allons continuer avec la texture crée précédemment, mais avec un masque crée par nos soins. La procédure de création d'un masque est identique à celle précédente. Il faut ensuite la sélectionner dans le menu ***Textures*** de l'outliner et charger le masque voulu (🎥 Vidéo 1 06:35).

Vérifiez que la bonne texture est sélectionnée dans le panneau ***Texture*** et que le mode d'application est ***Stencil*** dans les deux cas, afin de simplifier le processus (🎥 Vidéo 1 07:05).

On peut alors voir l'effet du masque sur l'aplat de la texture précédemment chargée (🎥 Vidéo 1 08:05) (figure 93).

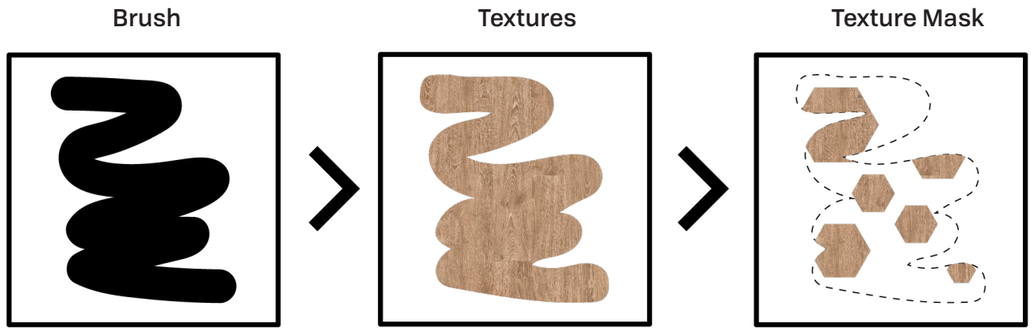
Texture baking

Le ***baking*** est une pratique essentielle dans la création de textures, mais se crée de plusieurs manières selon l'outil utilisé. Nous pouvons définir le ***baking*** comme le « processus de transfert de données d'un modèle à un autre ».

La texture ***NormalMap*** utilise le baking pour être crée, en générant une texture permettant de créer des reliefs à partir d'une différence entre deux modèles, l'un détaillé, l'autre simple.

Dans notre cas précis, le baking va nous servir à générer une texture sur l'UV déployé d'un modèle, à partir de textures pré-existantes dont le positionnement aura été géré par Blender. Cela nous permettra d'avoir une texture modifiable grâce aux outils vus précédemment de ***Texture Painting*** et utilisable dans un moteur de jeu.

Pour donner un exemple, nous allons travailler sur une petite scène et plus précisément la texture de la route (figure 94).



93. Schéma explicatif de la hiérarchie des fonctions

La première étape consiste à créer un matériau pour la route, en cliquant sur l'onglet ***Material***, puis ***New*** (🎥 Vidéo 2 00:15).

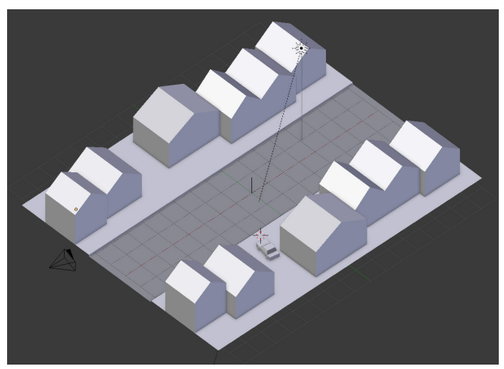
Cela nous donne accès au contrôle du matériau via le système de nodes expliqué auparavant.

La première étape est d'utiliser les éléments présents dans le ***Node editor*** de Blender pour plaquer la texture de base de notre route et l'ajuster à celle modélisée.

Nous avons donc besoin d'une texture, d'un node qui permettra de régler les dimensions, l'échelle et la multiplication de la texture sur le modèle, le ***Mapping*** node et d'un node établissant la manière dont la texture est ajusté, le ***Texture coordinate*** node. La texture peut être glissée/déposée depuis la navigateur de fichier (🎥 Vidéo 2 00:40).

Tous ces éléments reliés vont permettre de commencer à visualiser la texture sur le modèle.

Le node ***Mapping*** permet de régler et d'ajuster la position de la texture (🎥 Vidéo 2 01:40).



94. Scène Blender

Une fois cette étape effectuée, nous pouvons importer les autres textures de références gérant les effets, et les connecter aux bonnes entrées de notre shader (🎥 Vidéo 2 02:20).

Dans notre cas, les textures sont la diffuse map, la gloss map, reflection map et normal map.

Comme évoqué précédemment, sans convention exacte de tous les types de textures, il faut s'assurer de leurs rôles avant de les utiliser, sous peine d'avoir des effets visuels faux.

Dès lors, la reflection map doit être connectée à l'entrée ***specular*** puisqu'il est question de gestion des effets de la lumière, tandis que la gloss map doit être connectée à l'entrée ***roughness***, puisque cette texture a comme objectif la gestion de la rugosité. Toutefois, la gloss map doit être inversée, pour être conforme aux exigences du shader.

Enfin, la normal map requiert un node en plus, le normal map node, qui sera ensuite connecté à l'entrée normal du shader (🎥 Vidéo 2 03:25).

Une fois ces opérations effectuées et la texture ajustée, nous pouvons déployer l'UV et commencer le premier bake afin d'obtenir la texture de base, dimensionnée selon nos besoins.

L'UV déployée doit garder des proportions similaires à celle du modèle, afin d'éviter les étirements de la texture plus tard (🎥 Vidéo 2 05:00).

Après avoir correctement proportionné l'UV, il faut créer la texture qui accueillera notre bake. Évidemment, les dimensions de la texture doivent fournir des proportions approchant celles de l'UV (🎥 Vidéo 2 05:35).

Afin de baker la texture, il est nécessaire d'effectuer quelques réglages dans le matériau de la route. Pour que Blender puisse effectuer correctement son bake du matériau, il faut

créer un node ***Image texture*** pointant vers la texture que nous avons créée (🎥 Vidéo 2 05:40).

Ensuite, afin de gagner du temps, nous allons chercher à baker uniquement l'image ajustée selon le node ***Mapping*** sans aucun effet appliqué, puisque ce sont l'ensemble des couches de textures et le shader qui en ont le rôle.

Pour cela, nous allons utiliser un addon, le ***Node Wrangler*** (fiche tuto sur le Node Wrangler) pour outrepasser le shader, et n'afficher qu'une texture sur notre route, sans aucun effet appliqué (🎥 Vidéo 2 06:05).

L'option de bake se situe dans l'onglet ***Render***, au bas de la section. Après avoir sélectionné le node ***Image texture***, qui sera le support de notre bake, dans le matériau, nous pouvons lancer le bake en cliquant sur le bouton. Une barre de progression s'affiche normalement en haut pour indiquer l'avancée. N'oubliez pas de sauvegarder l'image (🎥 Vidéo 2 06:55).

Cette étape est à répéter pour chaque type de texture.

Une fois les textures générées, nous allons pouvoir utiliser l'outil de texture painting afin d'ajouter des éléments et des effets au modèle.

Pour cela, nous allons créer une nouvelle texture, nous permettant de « peindre » des détails supplémentaires, que nous mixerons à la texture via un node dédié (🎥 Vidéo 2 13:50).

Après avoir créé une nouvelle texture et utilisé l'outil ***Texture Painting*** pour rajouter des éléments, nous allons utiliser le node ***MixRGB*** pour fusionner les textures et les baker afin de les exporter vers un moteur de jeu (🎥 Vidéo 2 19:40).

Ce node permet de connecter en entrée deux images et de les superposer de plusieurs manières, de manière très similaire aux modes de fusion de Photoshop. Cela va déterminer la manière dont les pixels se mélangent, d'une image à l'autre.

Dans notre cas, nous allons superposer l'image bakée de la route avec l'image comportant des détails peint via l'outil texture painting. Et dans ce cas particulier, le mode de fusion ***Screen*** semble le plus adapté (🎥 Vidéo 2 19:48).

Créer une texture à part pour rajouter du détail a plusieurs avantages, notamment celui de pouvoir la modifier sans altérer la texture d'origine, mais aussi de pouvoir combiner les outils de Blender afin de fusionner cette texture avec toutes les autres pour simuler les différents effets face à la lumière (🎥 Vidéo 2 20:58).

De plus, utiliser le node ***MixRGB*** nous permettra de baker la texture avec ses détails directement, en outrepassant le shader via le ***Node Wrangler***.

Chaque cas est particulier, et nécessitera un paramétrage particulier selon la texture d'origine sur laquelle nous cherchons à appliquer du détail.

L'une des limites de Blender est de ne pouvoir appliquer, comme Substance Painter, toutes les couches d'un « matériau » simultanément, afin de rester au plus proche des textures d'origines gérant les effets. Il est nécessaire d'appliquer soi-même chaque couche, ou d'essayer de mettre à profit le système de nodes pour s'en passer.

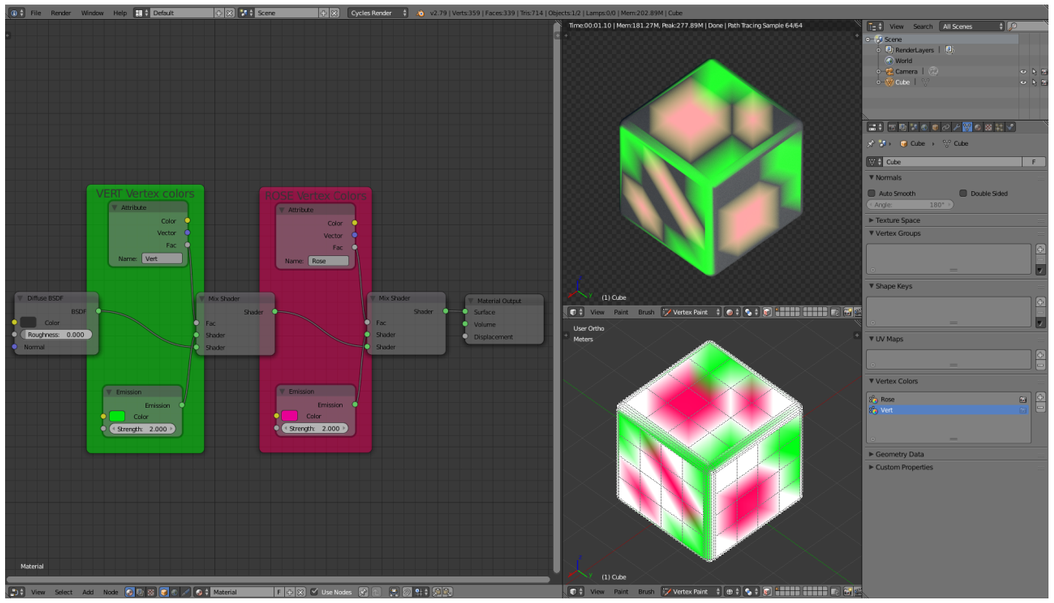
Vertex painting

Une autre méthode existe pour placer des textures sur un modèle, différente par la manière de localiser la texture. Le ***Vertex Paiting*** pouvant être traduit par « Peinture sur vertices » consiste à utiliser les vertices comme indicateurs de textures, en créant plusieurs ***Vertex Maps*** définissant où les textures sont appliquées.
L'inconvénient majeur de cette méthode est la nécessité d'avoir un très grand nombre de vertices et donc de polygones, si l'on veut être précis. Toutefois, cette méthode peut être combinée avec les techniques plus traditionnelles d'UV Mapping afin de détailler certains points spécifiques (figure 95).

Conclusion

Le jeu vidéo comme industrie a la capacité d'absorber et de s'approprier les nouveautés de manière très efficace. Nous venons de voir comment l'utilisation de quelques textures peuvent permettre de créer des matériaux et des effets visuels variés, adaptés à plusieurs approches sensibles.

Les moteurs de jeux ont la capacité d'utiliser ces moyens pour rendre en temps réel un environnement, un objet et le flux de travail PBR va permettre d'utiliser tout ce qui a été créé pour l'intégrer dans une scène parcourable en réalité virtuelle, offrant de nouvelles possibilités, non seulement de conception, mais aussi d'interaction.



95. Vertex Paint

FICHE RAPPEL TEXTURES

COLOR MAPS

Gestion des couleurs

SPECULAR MAPS

Comportement de la lumière sur le matériau

BUMP MAPS

Relief du matériau

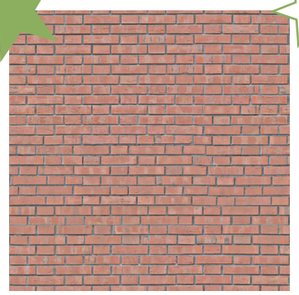
LIGHT MAPS

Points lumineux du matériau

Indispensable

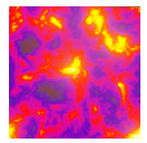
Parfois nécessaire

Consommatrice de ressources



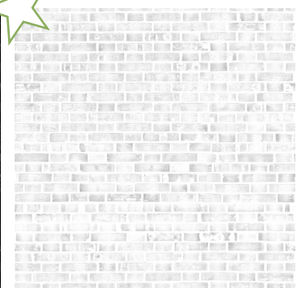
Diffuse / Albedo / Color

Detail



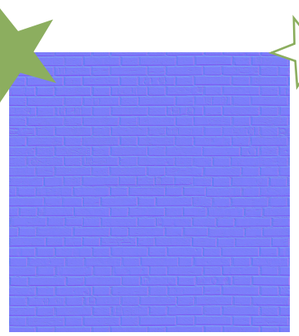
Gradient

Specular / Reflection

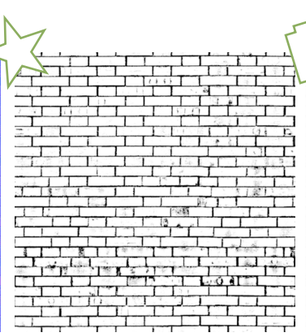


Roughness / Gloss

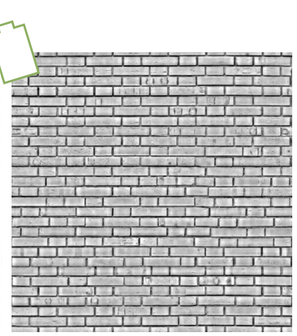
Metallic



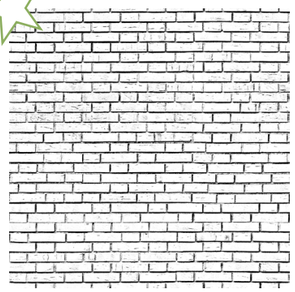
Normal



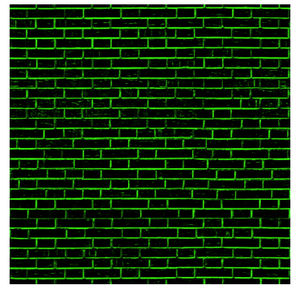
Bump



Displacement / Height



Ambient occlusion



Emission / Light

PARTIE 3 :
APPLICATION AUX AMBIANCES

Les différentes méthodes et outils énoncés auparavant sont utilisés dans l'industrie du jeu vidéo, capable de créer différents univers et atmosphères à partir de bases communes. Ces bases sont le travail du modèle, de la texture et du shader, qui vont donner un premier sens à l'univers qu'ils cherchent à créer et aux éléments qui en font partie.

Nous allons maintenant voir comment ces bases s'intègrent dans un moteur de jeu, qui va nous permettre de passer d'une image en 2D, à un environnement en trois dimensions et les contraintes qu'elles impliquent.

Définition

Le concept de base d'un moteur de jeu est simple à comprendre. Il s'agit d'une plateforme chargée d'effectuer des tâches en lien avec le jeu, tels que le rendu à l'écran, des calculs physiques ou encore la gestion des interactions entre le joueur et le monde.

Cela permet aux développeurs de se concentrer sur le sens qu'ils cherchent à donner à leurs réalisations, tandis que le moteur est capable de gérer les fonctions de bases, qu'il sera tout de même nécessaire de paramétrer[1].

Un moteur fonctionne selon un système de couches, allant du matériel de la machine à l'image affichée à l'écran. Le matériel ou ***Hardware*** est contrôlé par les pilotes, dont le rôle est de communiquer les informations, gérer les ressources et traduire les ordres des couches supérieures. Puis, le système d'exploitation entre en jeu, et orchestre les différents programmes sur la machine, en laissant la liberté d'accès ou non à certaines fonctions de la machine selon les besoins du programme. Ensuite apparaissent les libraires, qui sont l'ensemble des fonctions mathématiques et les algorithmes permettant de gérer nombre d'aspects du moteur. Les éléments des libraires sont pris en charges par la plateforme de gestion du jeu, faisant en sorte qu'il fonctionne quel que soit le support. À partir de cette étape, les fonctions principales du moteur apparaissent, avec le système central comprenant les fonctions du jeux, couplés avec le gestionnaire de ressources en charge de gérer les textures, modèles 3D, sons, etc. pour finir par le moteur en lui-même et son code, permettant d'exécuter chacune des instructions créant l'image à l'écran, c'est à dire la frame[2] (figure 96).

Classification

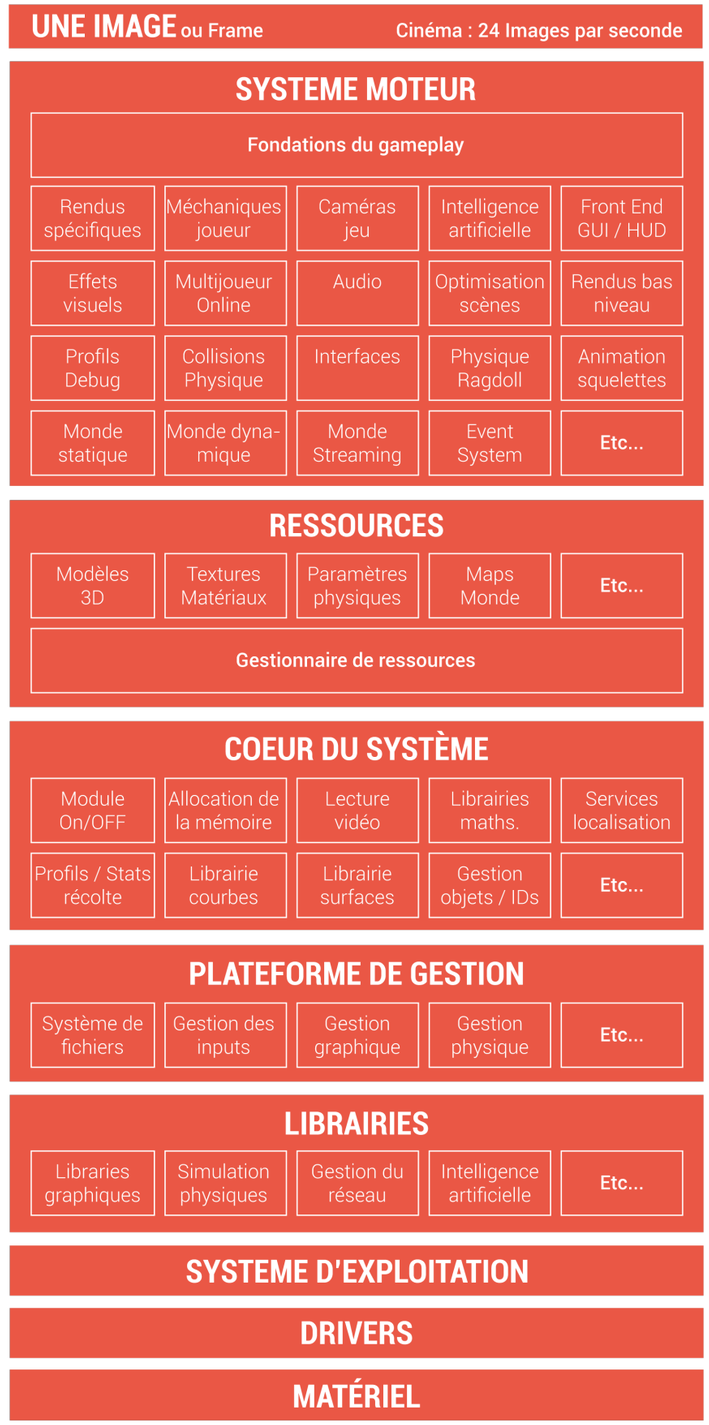
Il existe plusieurs moteurs de jeu, classés en trois niveaux suivant leur complexité d'utilisation.

Le premier niveau est appelé ***Roll-your-own*** et est considéré comme le plus bas niveau d'utilisation. Le moteur n'existe pas et c'est au développeur de le créer en utilisant les différentes librairies existantes pour gérer tous les aspects de son jeu, c'est-à-dire le rendu, le son, l'intelligence artificielle, etc. Ce cas est de plus en plus rare, de par l'ouverture des moteurs au grand public et le travail effectué sur l'accessibilité par leurs créateurs.

Le second niveau est appelé ***Mostly-Ready game engines*** et propose une grande variété d'outils à disposition du créateur, tout en laissant une marge de liberté sur le code afin de l'adapter à un jeu particulier. Nous pourrions citer le ***Cry Engine 3*** de Crytek ou l' ***ID Tech*** de ID Software (figure 97).

Enfin, le dernier niveau est le plus accessible et est appelé ***Point-and-click engine***. Il s'agit de moteurs tels que Unity3D ou Unreal Engine plus récemment. Beaucoup d'options sont disponibles sans effectuer de code et l'ouverture à la communauté permet d'utiliser le travail d'autres amateurs pour soi-même[3] (figure 98).

L'ouverture récente des moteurs au grand public n'est pas que purement altruiste.



96. Schéma moteur

1. HISTORY AND COMPARATIVE STUDY OF MODERN GAME ENGINES

2. Pause Process : Les moteurs
3. A brief guide to game Engines, David Parsons

En effet, le contexte économique actuel du jeu vidéo et l'augmentation des ventes dématérialisées a ouvert de nouvelles perspectives dans les ventes du jeu vidéo, laissant une place à des créateurs indépendants n'ayant pas besoin d'éditeurs.

Dès lors, libérer son moteur offre une plus grande visibilité auprès du public, et une potentielle publicité à long terme s'il est utilisé. C'est notamment le cas pour l'Unreal Engine, qui reste sans doute le moteur le plus puissant et le plus beau du marché, mais reste légèrement restrictif dans le type de jeu qu'il est possible de créer[4].

Dans notre cas, nous utiliserons un moteur de dernier niveau, très accessible, pour construire un environnement d'expérimentation architectural.

Un moteur est composé de plusieurs modules, comme nous l'avons évoqué auparavant et son utilisation est possible grâce à deux éléments, l'API et le SDK. L'API est l'abréviation de ***Application programming interface*** et est donc l'interface du logiciel qui va nous permettre de gérer le moteur par le SDK, ***Software Development Kit*** ou la collection d'outils, bibliothèques et sous-SDK permettant de construire un jeu[5].

Pour l'architecte

L'intérêt pour les architectes d'utiliser ces outils est de permettre une simplification du processus de travail et de développer de nouvelles manières de montrer le projet, au travers d'outils prêts à l'emploi du jeu vidéo, avec un grand nombre de possibilités d'interactions.

De plus, l'intégration d'une 3ème dimension permet de mieux comprendre l'espace et le volume du projet, autant en phase de conception qu'en démonstration pour un client.

Cette tendance est confirmée par une étude effectuée sur des entreprises spécialisées dans le rendu d'architecture, évoquant que le second moteur le plus expérimenté aujourd'hui est l'Unreal Engine, soit un moteur dédié au jeu vidéo[6].

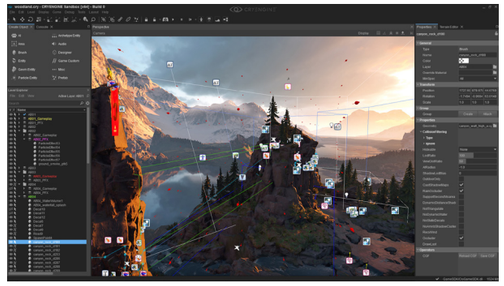
Toutefois, pour un architecte, la difficulté d'adaptation et d'apprentissage est bien réelle et c'est pourquoi il est difficile de faire un changement aussi direct entre des outils 2D et 3D. De plus, les moteurs de jeux ne sont pas encore totalement adaptés aux modèles architecturaux utilisés. Actuellement, il n'y a aucune compatibilité entre les formats CAO ou IFC et les moteurs.

Les moteurs ont la capacité de créer des environnements divers et de produire des styles et ambiances différentes pour une scène, et cela pose la question du rendu en tant que tel et son ancrage dans la réalité.

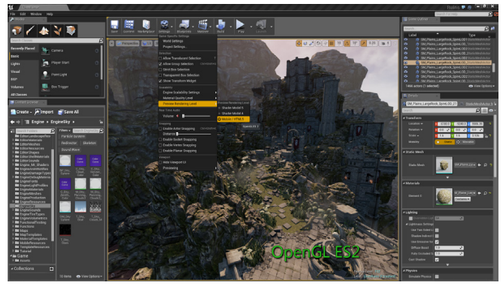
Mais tout comme l'image, le jeu vidéo peut mentir et donner de fausses informations. L'univers virtuel permet le jeu avec la profondeur, changer les règles que l'on croit connaître, le tout pendant que le spectateur est immergé dans l'environnement. Un exemple notable est le jeu ***Museum of Simulation Technology***, développé par Pillow Games, dont le but est de passer des salles à énigmes en jouant avec la perception de la perspective. Par exemple, en agrippant un objet devant soi, pour le bouger dans la perspective de manière à ce que le panneau d'issue de secours se transformer en mur au bout du couloir (figures 99 & 100).

L'ajout d'une dimension temporelle et d'une liberté pour le joueur implique aussi qu'il peut voir ce qu'il veut, tant que le jeu le permet. Et parfois, même lorsque le jeu ne le permet pas. Certains joueurs ont fait de leur activité principale le fait de « casser » un jeu pour repousser les limites prévues par le développeur. Le ***speedrun*** est une activité consistant à terminer un jeu le plus vite possible, quitte à utiliser des moyens non prévus pas les développeurs, en abusant de a physique ou des erreurs et oublis de programmation. De cette manière, il n'est pas rare pour un de ces joueur de passer hors des limites prévues par le jeu et d'en voir des éléments qu'il n'est pas possible de voir par des moyens conventionnels (figure 101).

C'est pourquoi nous allons nous concentrer sur les possibilités qu'offre le moteur de jeu dans une découverte architecturale, en commençant par voir en quoi le moteur de jeu prolonge le travail effectué sur le modèle et la texture expliqué auparavant et les fonctionnalités propres au moteur de jeu influant sur la perception de l'environnement.

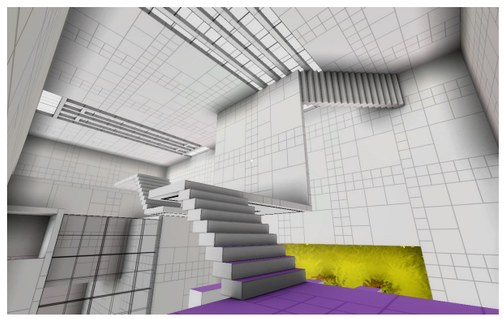


97. Cry Engine

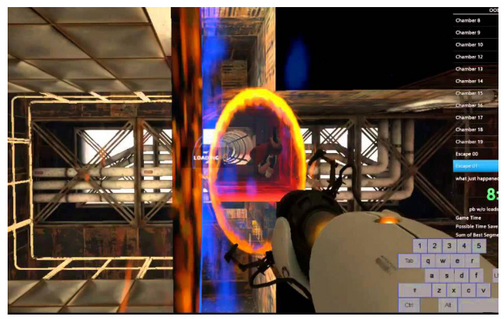


98. Unreal Engine SDK

99. Pillow Castles



100. Manifold garden



101. Speedrun sur le jeu vidéo Portal

4. Pause Process : Les moteurs
5. What is a game engine, Jeff Ward, 2008
6. 2016 Architectural visualization rendering engine survey, Jeff Mottle, 30 Novembre 2016

La profondeur

L'image limite la visualisation à deux axes X et Y, nous sommes face à un point de vue orienté, ne nous offrant aucune liberté de parcours et de découverte. Le moteur de jeu crée un autre axe pour générer le volume, l'axe Z.

Intégrer une 3ème dimension à un monde et le rendre parcourable pose plusieurs questions. Contrairement à une image, on ne choisit pas ce que le joueur regarde, le joueur entre dans le cadre, regarde où il veut et y passe le temps qu'il souhaite[1].

L'environnement est alors porteur de sens et tous les éléments d'une scène composent une architecture numérique, créant un langage visuel nous donnant les règles du lieu. On peut alors se servir de ce qu'on l'on connaît du monde pour mieux nous divertir[2].

La caméra

Ce qui est vu est transmis par ce qu'on appelle une caméra, le palier principal du joueur vers son univers virtuel. Tout comme un rendu fixe, elle a un rôle très important à jouer dans la perception d'un lieu et le sens à donner. Elle sera différente selon le jeu, prenons le cas d'un jeu de tir, il existe des FPS (Jeux de tir à la première personne) (figures 102 & 103) et TPS (Jeux de tir à la troisième personne) (figure 104), dont la différence est la position de la caméra.

La caméra a la première personne dans un jeu de tir est souvent faussement placée, puisque tout ce que nous cherchons à voir, c'est l'arme utilisée et la cible, mais elle renforce l'impression d'immersion. Au contraire, une caméra à la troisième personne nous laisse plus de liberté pour observer notre environnement en proposant un recul sur le personnage.

Tous ces choix se font en fonction du besoin du jeu et de la volonté des développeurs. Dans notre cas, nous mettons en avant les différents cas de caméra adapté à une manière de jouer, mais l'influence du cinéma et de la photographie est bien visible lorsqu'il s'agit de cadrer une scène ou une cinématique en particulier. De plus, les développeurs recherchent à rendre lisible l'action du jeu, cela requiert quelques fois une adaptation manuelle de l'emplacement de la caméra afin de figurer le meilleur angle.

Le premier jeu ***Resident Evil*** (1996) base même son principe là-dessus, en n'utilisant que des caméras fixes, afin de contraindre le joueur dans son environnement, et provoquer une tension permanente (figure 105). Les raisons sont multiples, la première étant que le jeu était orienté action/horreur et la caméra répondait aux exigences du jeu. La seconde étant plutôt matérielle, car pour l'époque, créer une caméra mobile dans un environnement en trois dimensions était encore un exercice difficile.

Les médias s'inspirent de l'un et de l'autre dans leurs recherches visuelles et il n'est pas rare de voir des références explicites à des films dans les jeux vidéos. (figures 106 & 107)

Options en temps réel

Concernant la caméra, les moteurs ont d'autres spécificités, ce sont les effets appliqués en temps réel. En effet, sur une image fixe, les effets sont calculés une seule fois, tandis qu'un moteur se doit de calculer chaque effet, image par image, selon la fréquence de rafraîchissement du jeu. Le fait de devoir, comme au cinéma, afficher un grand nombre d'images par seconde, nécessite des ajustements et des traitements pour l'affiner.

Nous allons catégoriser les options graphiques en commençant par les options corrigeant les défauts à l'écran.

102. Exemple de FPS : Counter Strike Go

104. Exemple de TPS : Splinter Cell Blacklist

107. Comparaison Metal Gear Solid V / Mad Max

103. Illustration humoristique de la position de la caméra

105. Camera fixe - Resident Evil I

106. Vidéo d'introduction de Metal Gear Solid Ground Zero

1. Dishonored 2 ou l'art au service du jeu, 30 Novembre 2016
2. Game Architect, Bits ARTE

Corrections

L'un des problèmes les plus fréquents est l'***Aliasing***, c'est à dire l'effet d'escalier dessiné par les bordures d'un objet non verticales. Cela apparaît car nos écrans ne peuvent afficher l'image de manière assez grande pour empêcher de « casser » la géométrie du modèle.

Le phénomène est le même lorsqu'on essaie de réduire la dimensions d'une image, les pixels apparaissent de plus en plus, et il est nécessaire de légèrement flouter le tout pour les faire disparaître. C'est exactement cette méthode qui est appliquée pour ce qu'on appelle l'***Anti-Aliasing***. Il existe plusieurs types d'anti-aliasing, certain propriétaires, d'autres non, mais le principe générale est de flouter la zone pixelisée pour atténuer l'effet (figure 108).

Un autre problème est la gestion des textures selon la distance et l'angle de vue du joueur. En effet, contrairement à une image, le jeu vidéo permet au joueur de s'approcher d'un objet, même si sa texture est de très basse résolution. A contrario, il peut aussi s'en éloigner, alors que la texture très détaillée est chargée inutilement.

Pour résoudre ce problème, il a été créé le filtrage bilinéaire et trilinéaire, une méthode qui va calculer la distance à laquelle la caméra se situe et gérer la texture pour l'afficher de la meilleure manière. Mais cette méthode ne prend pas en compte l'angle de la texture par rapport à la caméra, ce qui crée des artefacts visuels lorsque les faces de l'objet sont presque perpendiculaires à l'origine de la caméra (figure 109).

Le filtrage anisotropique est en charge de prendre en compte l'angle de l'objet par rapport à la caméra, pour calculer correctement la manière d'afficher à l'écran les pixels. Un exemple pourrait être un brique face à la caméra. Si cette brique reste exactement face à la caméra, recalculer la texture selon la distance est aisé. Mais si la brique tourne sur elle-même, les faces ne sont plus des rectangles sur l'écran, c'est pourquoi le filtrage doit être réalisé en fonction de l'objet également (figure 110).

Effets

Puis, il existe un grand nombre d'options définissant les caractéristiques graphiques d'un jeu. Les plus connues et utilisées sont le Bloom, l'Ambient Occlusion, la profondeur de champ ou plus récemment l'HDR.

Le bloom augmente la luminosité des lumières et y ajoute un halo lumineux pour simuler l'éblouissement lorsqu'on regarde une source de lumière directement (figure 111).

L'ambient occlusion est un effet appuyant les éléments dans un décor, en ajoutant des ombres là où deux surfaces entrent en contact ou deux éléments se rencontrent. Cela permet de créer un faux effet de relief dans le décor, sans raccord avec la lumière de la scène (figure 112).

Le ***Depth of Field*** ou profondeur de champ en français, est, comme en photographie, un effet de flou appliqué aux éléments qui ne sont pas regardés. Si l'on regarde un objet proche, le fond est flou et vice-versa. Le but est de rajouter un effet similaire à un film ou une photographie (figure 113).

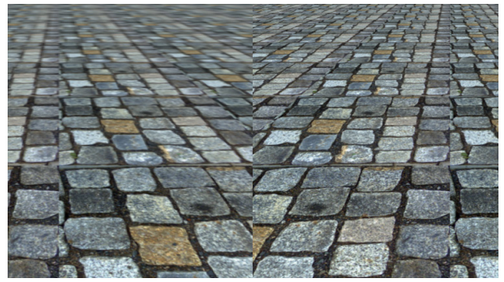
Il existe encore un grand nombre d'effets ayant chacun un rôle sur l'image, la difficulté étant de les énumérer puisqu'il en existe un très grand nombre, parfois libre, mais parfois propriétaire des fabricants de certains modèles de carte graphiques.

Nividia, par exemple, propose des solutions clés en main pour certains effets dans les jeux, à utiliser pour les développeurs. Nvidia a proposé il y a deux ans un algorithme ***HairWorks*** pour la simulation des cheveux et des poils sur un objet.

Dans notre cas, nous nous limiterons aux effets disponibles sur le moteur que nous allons utiliser, Unity 3D 5.

108. Aliasing / Anti-Aliasing

109. Filtrage bilinéaire / trilinéaire



110. Filtrage anisotropique

111. Bloom

112. Ambient occlusion

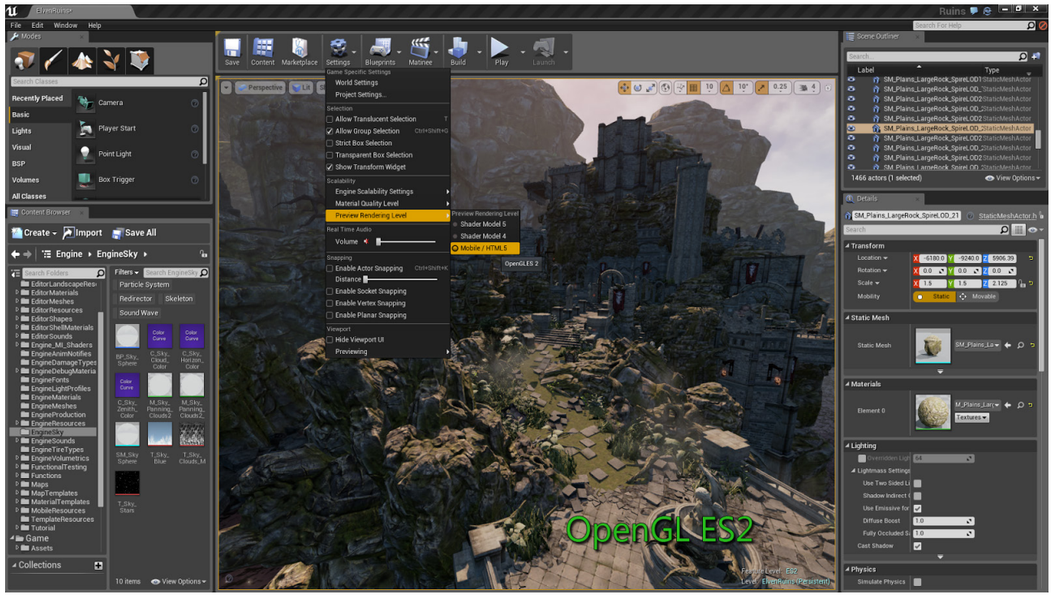
113. Profondeur de champ

Utiliser le moteur

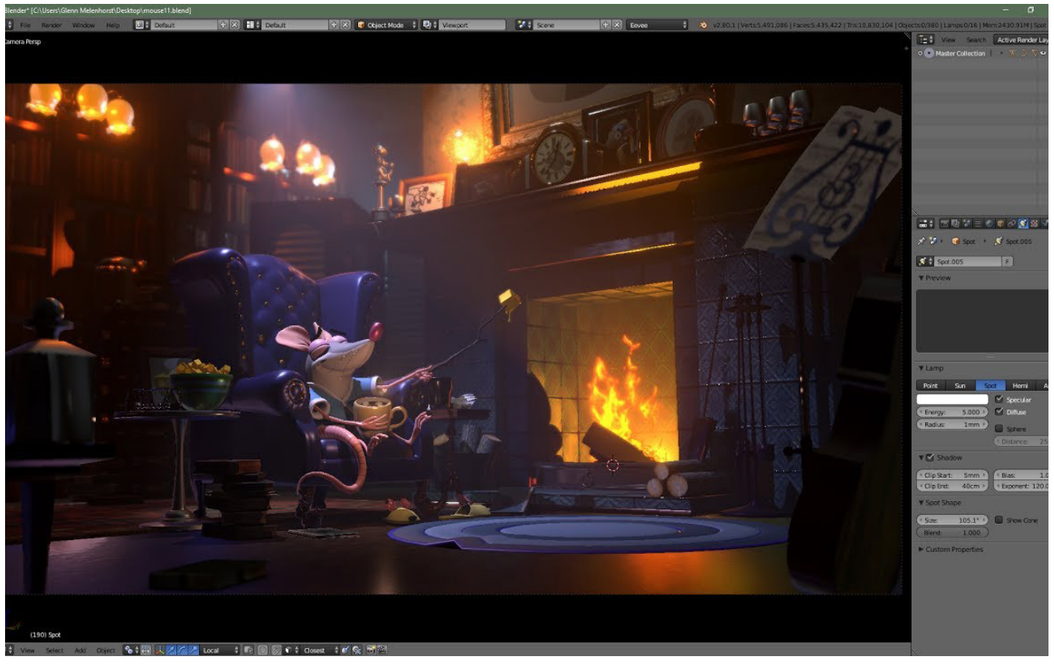
Le travail de modélisation, de texture, d'UV Mapping et l'utilisation des méthodes PBR vont nous permettre de passer très facilement sur des moteurs de jeux récents.

Les deux plus gros moteurs de jeu gratuit et accessibles actuellement sont l'Unreal Engine 4 et Unity 3D 5. Nous allons utiliser Unity pour son accessibilité, ses fonctions d'échanges d'outils et sa communauté permettant de facilement trouver du contenu et de l'aide (figure 114).

Blender comprend lui aussi un moteur de jeu, sobrement appelé ***Game Engine***, mais n'est plus mis à jour et présente des retards par rapports aux technologies actuelles. Toutefois, une nouvelle mouture du moteur appelée ***EEVEE*** ou ***Extra Easy Virtual Environment Engine*** est en phase de test et prend en charge les technologies PBR, un petit nombre d'effets tout en prenant en charge le rendu en temps réel. Toutefois, il n'existe aucun outil permettant de faire plus que visualiser un modèle et il reste encore très sommaire (figure 115).



114. Unreal Engine 4



115. Moteur de rendu en temps réel EEVEE

Capacité d'adaptation

Les moteurs sont alors des outils dont la capacité à simplifier l'exécution et le développement d'un jeu permettent de se concentrer sur le jeu, ses intentions et volontés pour créer un projet complet.

Il est possible de créer de multiples ambiances, univers et atmosphères au sein d'un même outil, en réutilisant des éléments et en les réinterprétant. Il existe un grand nombre d'exemples pour le moteur Unity, réussissant à créer des jeux en 2D, en 3D avec des points de vues, des histoires et des gameplay totalement différents.

Nous pourrions citer des jeux tels que ***Rust***, un FPS dans un monde ouvert, laissant le joueur libre de créer ses objets pour survivre, ***Firewatch*** (figure 116), un jeu aussi à la première personne, centré sur l'histoire du personnage que l'on incarne, dans un style graphique coloré et non réaliste. Dans d'autres registres, tels que la simulation spatiale, il existe aussi ***Kerbal Space program*** (figure 117), jeu de gestion d'un centre spatial avec création de fusées, missions d'analyse et conquête de planète. En tant que jeu 2D, le jeu ***Cuphead*** (figure 118) reprend le style graphique des premiers dessins animés Disney pour l'adapter à un jeu de plateforme exigeant.

Le moteur permet l'export sur différents supports et nous pouvons alors parler de ***Monument Valley*** (figure 119), jeu sorti sur iOS, axé sur la réflexion et des puzzles en vues isométriques.

Enfin, plus rares, il existe aussi des visualisations de projets, par exemple une démonstration de maisons en rondins de bois réalisée par une entreprise de créations de démos interactives pour l'architecture, offrant à l'utilisateur la capacité de changer des éléments d'une pièce, les sols, plafonds, couleurs et de visualiser en direct un futur produit (figure 120).

Créer des scénarios

Cette flexibilité de création est adaptée pour la visualisation architecturale, qui permet de créer plusieurs ambiances et atmosphères à partir d'un seul et même outil, permettant d'adapter son rendu à la personne qui le regarde.

Cela nous rapproche de la question du sens à donner à une représentation et au public auquel elle s'adresse. Un même type de jeu peut être décliné en plusieurs versions avec un ***gameplay*** similaire, mais différent par le public visé et la représentation qu'il en est faite.

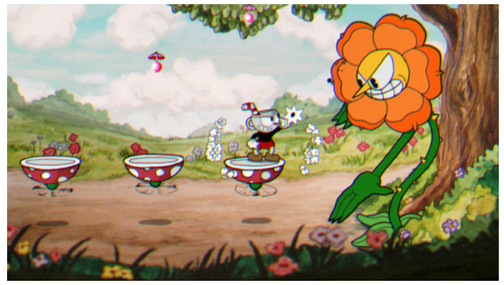
Plusieurs jeux se sont déclinés de cette manière, nous pouvons par exemple citer la saga des ***Age of Empires***, jeu de stratégie dont la première itération a été publié en 1997. Il s'agit d'un jeu de stratégie en temps réel, dont le but est de faire évoluer une civilisation antique sur plusieurs millénaires. La première mouture du jeu revêt un style très détaillé et volontairement réaliste pour rester dans l'ambiance sérieuse du titre (figure 122). Pour autant, le jeu s'est décliné en une version beaucoup plus ouverte au public, appelée ***Age of Empires Online*** en 2011 et cela a nécessité des changements dans l'aspect graphique afin d'attirer un plus grand nombre d'utilisateurs ? Dès lors, le style visuel est devenu beaucoup plus coloré, en abordant un léger effet cartoon, abandonnant le réalisme historique recherché auparavant (figure 123).

Nous pouvons aussi présenter brièvement l'exemple de la saga ***Battlefield***, jeu de tir à la première personne, optant aussi pour un tournant en ligne en 2009 et passant d'un aspect réaliste à un jeu cartoon, gratuit et ouvert à une plus jeune tranche d'âge (figure 121).

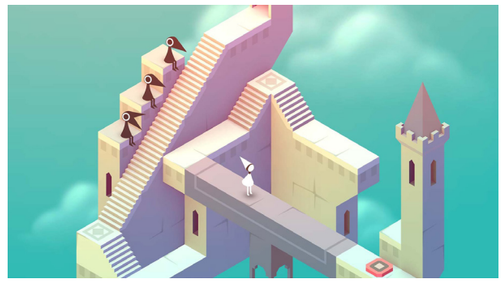
L'utilisation d'un moteur offre des possibilités, tant dans la perception d'un environnement en trois dimensions, que dans l'interactivité que peut offrir une scène, avec la prise de conscience du volume d'un espace et son appropriation avant sa construction.

116. Firewatch

117. Kerbal Space Program



118. Cuphead



119. Monument Valley

120. Hirsh Log Homes Visualization

121. Comparaison : Battlefield Heroes / Battlefield 1

122. Age of Empire, 1997

123. Age of Empire Online, 2011

Histoire

Unity 3D est un moteur de jeu multiplateforme développé par Unity Technologies.

La première version du moteur a été lancée en Juin 2005, avec comme objectif de créer un produit abordable, offrant des outils professionnels pour le développement de jeux par des amateurs. L'inspiration majeure de leur travail est la simplicité d'utilisation que l'on pouvait retrouver dans d'autres logiciels, avec une interface axé autour du système de « Glisser / Déposer »[1] (figure 124).

A titre de comparaison, l'Unreal Engine ne sera disponible au téléchargement que quatre ans après, en 2009[2].

Au fil des versions, la priorité des développeurs a été de mettre en avant la capacité du moteur à exporter sur différentes plateformes, en adéquation avec la prolifération des smartphones et des tablettes. Cela a permis d'étendre encore un peu plus la base d'utilisateurs et créer un écosystème de jeux basés sur ce moteur.

Fonctionnement

La fenêtre de Unity est composée de plusieurs éléments, gérant l'ensemble des scènes crée par l'utilisateur (figure 125).

Le moteur base son contenu sur un principe de ***Game Object***, comportant les options et les comportements destinés au jeu. Les fonctions sont intégrées au objets de la scène et ne sont pas codés à part. De cette manière, l'utilisateur peut mieux comprendre les relations entre tous les objets et décider des comportements de manière plus fine.

Il y a toutefois quelques éléments, notamment les options graphiques, de luminosité et d'éclairage, de fond ou encore les options réseaux qui sont gérées en dehors de ce système, car globale à toute la scène.

Tous les objets, modèles 3D, scripts, matériaux, sons, etc. sont accessibles dans la fenêtre ***Project***, et peuvent être déposés dans la fenêtre ***Scene*** via un simple glisser-déposer. Toutes les ressources nécessaires, de Unity ou de l'extérieur y sont visibles et peut être considéré comme le dossier du projet. Cette facilité d'utilisation est commune pour tout le logiciel et est utilisable pour tous les menus.

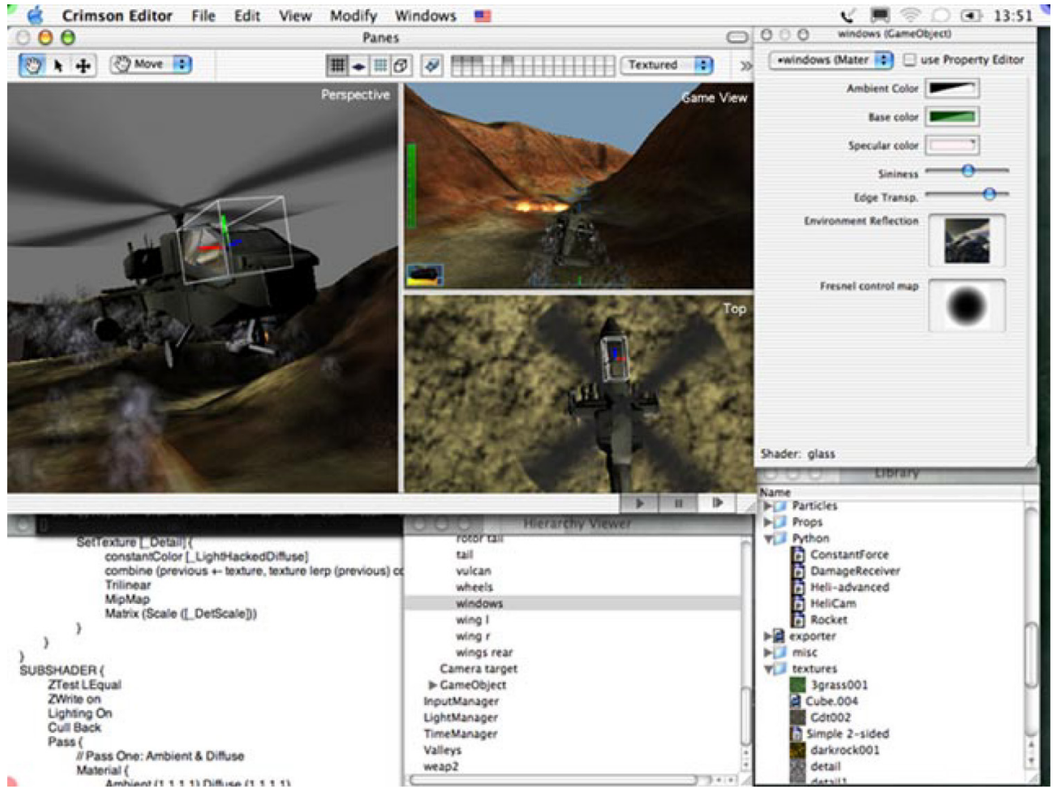
La fenêtre ***Hierarchy*** comprend tous les ***Game Object*** intégrés à la scène. Ils peuvent être renommés, dupliqués, et rangés. Un double-clic sur un des éléments effectuera un zoom sur celui-ci permettant de le repérer plus facilement.

L'***Inspector*** est la fenêtre de gestion des caractéristiques des ***Game Objetc***. Il s'agit du principal outil pour ajuster les propriétés de tous les objets et ressources de la scène, permettant d'y ajouter des scripts, shaders, etc. Un grand nombre de ces variables est sensible au système de glisser-déposer de Unity.

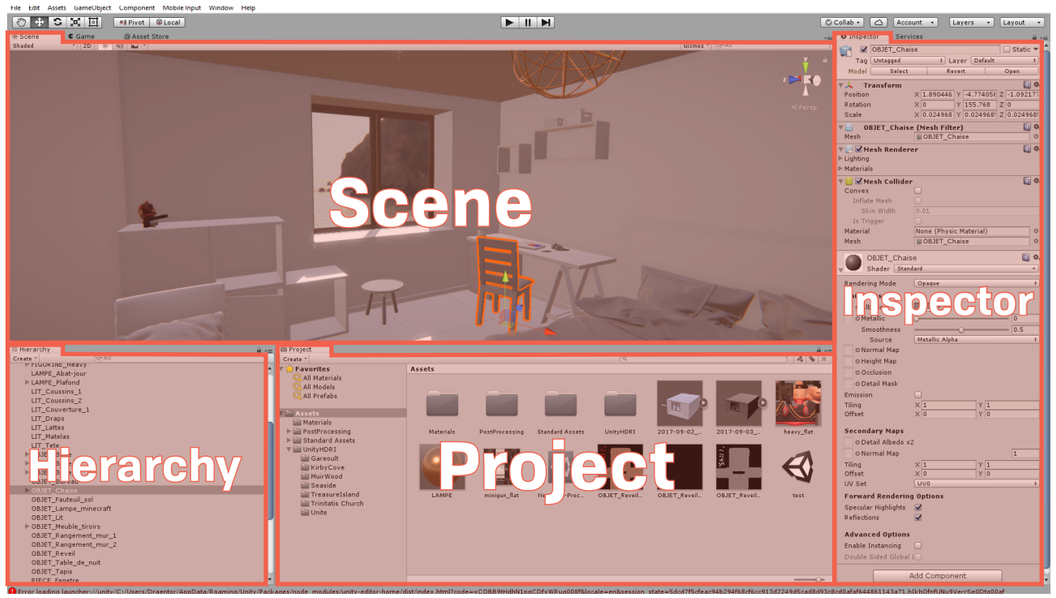
Les deux dernières fenêtres sont dédiées à la visualisation du projet, ce sont les fenêtres *Scene* et *Game*.
La fenêtre ***Scene*** permet de naviguer dans le projet, de sélectionner les objets et de les déplacer. La fenêtre ***Game*** est une prévisualisation en temps réel du jeu, selon le point de vue de la caméra, si existante. Elle applique aussi directement les effets et permet d'essayer la scène immédiatement.

Pourquoi Unity ?

L'objectif de Unity de s'adresser à des amateurs et de répondre à leurs attentes, aussi différentes qu'elles soient. Dès lors, Unity est capable de gérer différentes ressources et assure une compatibilité avec plusieurs logiciels, 3Ds Max, Maya, Cinema4D, Modo, etc. et Blender.



124. Unity Pre-Release



125. Fenêtres Unity

1. A History of the Unity Game Engine, John Haas
2. L'histoire de l'Unreal Engine, Opium Testing, 11 Février 2016

Usuellement, les moteurs demandent une conversion des objets au format .fbx, format de fichier propriétaire de Autodesk, pour pouvoir les utiliser. Unity est capable de reconnaître un fichier Blender et gère la conversion vers le moteur de manière automatique, en prenant en compte les matériaux crée et les différentes Uvs déployées. De plus, Unity est maintenant adapté à un flux de travail PBR, en gérant les différentes couches de textures par le biais de shaders.

L'ouverture du moteur à une aux amateurs du jeu vidéo a permis la création d'une grande communauté, échangeant leurs travaux et connaissances, notamment par le biais d'un ***Asset Store***, regroupant un grand nombre de ressources, gratuites et payantes, pour l'utilisateur. La popularité grandissante permet aussi une reconnaissance de plus grands acteurs du jeu vidéo et nous assurera la compatibilité avec d'autres équipements que nous verrons plus tard.

Enfin, dans la continuité du travail, la liberté de création permet une appropriation de l'outil afin de l'adapter à différents modes de rendus de la visualisation architecturale. Cette liberté ne se limite pas qu'à la production de jeux puisque récemment, Studio 4, une branche de France Télévisions, a créé un dessin animé de 13 épisodes appelé *Mr. Carton* uniquement avec Unity (figure 126).

De Blender à Unity

Utiliser un fichier Blender sur Unity requiert quelques vérifications et ajustements.

L'avantage d'utiliser un fichier Blender complet est de permettre de le modifier ultérieurement et de conserver les modifications en temps réel sur Unity. De plus, le moteur reconnaît les différents objets crées et est capable de les afficher correctement dans la hiérarchie de la scène.
Avant de glisser le fichier dans le dossier ***Assets*** du projet Unity, il sera nécessaire d'effectuer quelques vérifications.

A la différence d'un moteur de rendu 2D, le rendu en temps réel requiert des modèles 3D propres et les plus léger possible. Il existe plusieurs fonctions sur Blender permettant d'alléger un maillage et le nombre de vertices d'un modèle. Il faut aussi vérifier la direction des normales du modèle, car Unity n'affiche qu'un côté pour chaque face d'un modèle, le côté dans la direction de la normale.

Cela signifie alors qu'il est nécessaire, selon les cas, d'emballer le modèle avec un autre modèle, dont les faces sont tournées vers l'extérieur.

Toutes les fonctions et ***modifiers*** inhérents à Blender doivent être appliqués ou enlevés, Unity ne les reconnaîtra pas.

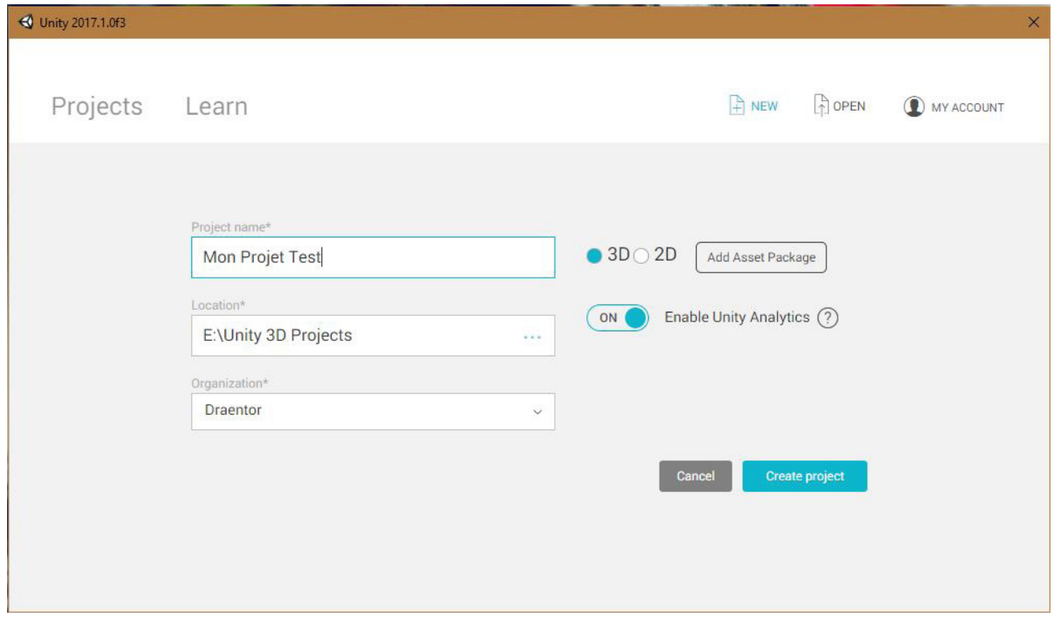
Toutefois, Unity sera en mesure de reconnaître les matériaux crées, mais ne sera pas en mesure de les afficher tel qu'ils sont configurés sur Blender. Comme tout moteur de jeu, il sera obligatoire d'utiliser les shaders et l'UV Mapping pour les textures de vos objets. Certains objets n'ont pas nécessairement besoin de texture dédiée et dans ce cas, tout comme avec Blender, il est possible d'appliquer un shader général, qui ne fera qu'un effet simple, sans relief ou texture.

Importer un fichier

Pour commencer, nous allons créer un nouveau projet Unity, choisir l'emplacement de destination, un nom et des assets complémentaires si besoin est. Tous les assets, de l'asset store ou de Unity peuvent être chargés ultérieurement dans le projet (figure 127).

Le projet crée, nous allons pouvoir importer notre fichier Blender complet directement dans le dossier assets crée par Unity. Pour cela, nous pouvons glisser/déposer notre fichier directement dans la fenêtre ***Project*** de Unity, en ayant pris soin de sélectionner le dossier ***Assets*** auparavant, ou copier/coller le fichier dans ce même dossier en utilisant l'explorateur

126. Mr Carton



127. Unity - Nouveau projet

de fichier du système d'exploitation (figure 128).

Une fois le fichier importé, ce qui peut prendre un peu de temps, il suffit de le glisser/déposer de la fenêtre ***Project*** à la fenêtre ***Scene*** dans Unity.

On peut d'ailleurs observer qu'il y a des erreurs, visibles dans la scène, et répertoriées dans la console au bas de l'écran. Il est listé une erreur de polygones superposé sur un objet et un trop grand nombre de vertices pour un autre objet. Ces erreurs peuvent être rectifiées sur Blender à posteriori, tandis que Unity sera en mesure de recharger et mettre à jour le modèle (figure 129).

Les matériaux crées sur Blender ont bien été importés, et nous allons pouvoir les relier aux objets correspondants. Pour cela, j'ai créé un nouveau dossier texture, dans le dossier ***Assets*** attribué à toutes les textures précédemment baké sur Blender. Comme auparavant, les textures ont été glissées/déposées depuis la fenêtre de l'explorateur.

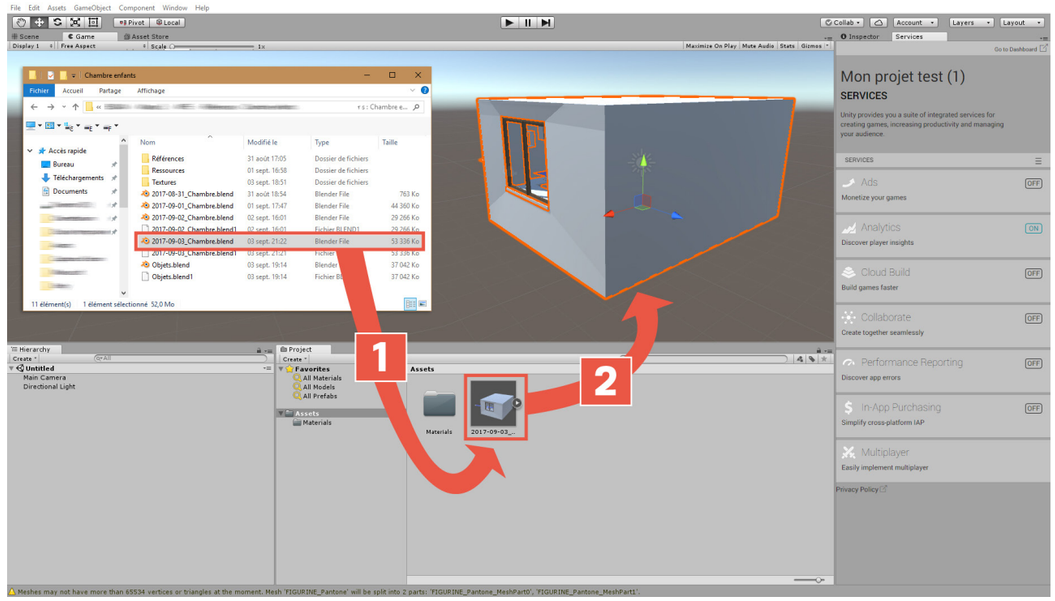
Les shaders de base de Unity gèrent différentes couches de textures, diffuse map, specular/metallic map, normal map, ambient occlusion map et emission map.

De base, il propose un shader ***Standard*** axé sur la différence entre dielectic et metallic d'un modèle. Dans notre cas, la plupart des objets seront dielectic, et devront alors changer de shader, en cliquant sur le bouton ***Standard*** et en variant la configuration à ***Standard Specular Setup***.

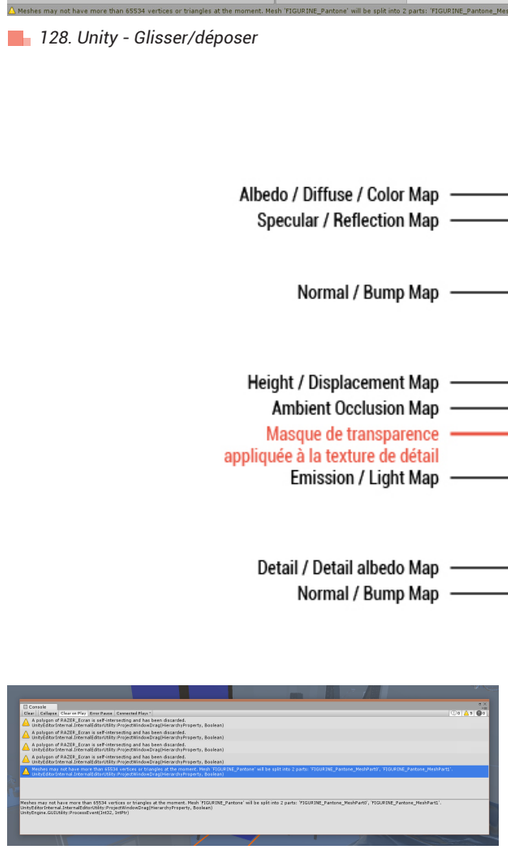
A partir de là, nous pouvons faire glisser nos différentes textures sur le shader. La normal map doit être marquée en tant que telle, ce que nous propose le logiciel en cliquant sur ***Fix Now*** (figure 130).
Le shader comprend plusieurs sections, en commençant par les textures nécessaires, c'est à dire la diffuse map, specular map et normal map. Ensuite, nous pouvons y glisser les textures gérant la déformation du maillage du modèle, la displacement map, accompagné de l'ambient occlusion map ainsi que la fonction ***detail mask***. Cela sert à appliquer une texture noir et blanc servant de masque à la texture de détail qui peut s'ajouter à la diffuse map et rajouter une couche de détail sur le modèle.

Il est aussi possible de cocher la case ***Emission*** permettant d'accéder à la fonction en charge de la texture ***Emission map***, dans le cas où le modèle présente une texture de ce type.

Enfin, la dernière section est consacrée aux textures de détail, avec une gestion plus fine des paramètres, comme la possibilité de la multiplier ou la déplacer sur le modèle.



128. Unity - Glisser/déposer



129. Unity - Erreurs console

130. Unity - Shader specular

Démarche

Un moteur de jeu et ses nombreuses possibilités en terme de création permettent de générer un grand nombre de situations de scènes différentes, avec des outils similaires. Dans le prolongement de l'étude des moyens du jeux vidéo, nous allons explorer en quoi le moteur pourrait permettre des visualisations d'un projet de manière plus complète, tout en offrant un degré de liberté permettant de modifier simplement les conditions et d'adapter le médium au spectateur.

Mise en application

Afin de générer plusieurs scénarios, nous allons créer une scène sous Blender, qui sera exportée vers Unity, en utilisant toutes les méthodes et outils expliqués auparavant.

Nous avons fait le choix de modéliser une chambre, avec comme objectif de la présenter en trois variantes basés sur la même modélisation, mais en se servant des fonctions de Unity et le flux de travail PBR pour générer trois ambiances différentes. Ce qui fait la force du travail des textures PBR, c'est l'adaptation de l'environnement à la lumière, et non le contraire. Dès lors, la scène réagira en conséquence des lumières établies et non le contraire.

Cette méthode est une proposition de construction d'une scène se référant aux méthodes utilisés dans le jeu vidéo, cherchant à être efficace dans l'utilisation des moyens, tout en permettant une flexibilité d'usages.

La première situation envisagée est une ambiance réaliste. Pour la réaliser, nous allons mettre en place plusieurs outils, de la gestion de la lumière par une image à la préparation de la caméra.

HDRi et Skybox

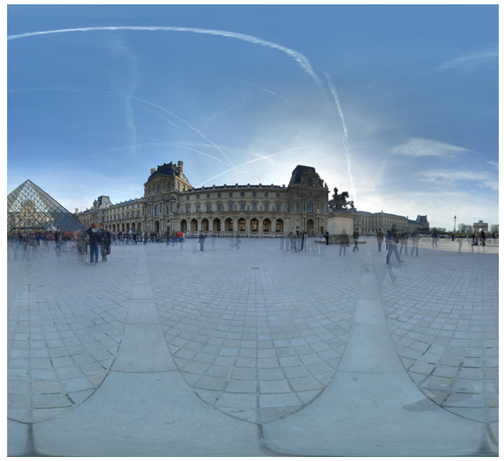
Lorsqu'il s'agit de fournir un environnement réaliste, un des points les plus importants est l'éclairage, qui aura un rôle très important sur la scène et les textures. Pour cela, nous allons utiliser une fonction de Unity, la ***Skybox***.
Une ***Skybox*** est une image qui servira à afficher l'environnement au-delà de la géométrie, et permet d'éclairer la scène selon cette image. Cette fonction existe aussi dans Blender, et permet d'utiliser des fichier dits ***HDRi***, abbréviation de ***High Dynamic Range Images***, pour éclairer la scène[1].

Ces images sont des clichés instantanés d'un lieu, capturés à 360°, permettant d'obtenir un éclairage très proche de la réalité de par la nature de la photographie (figure 131).

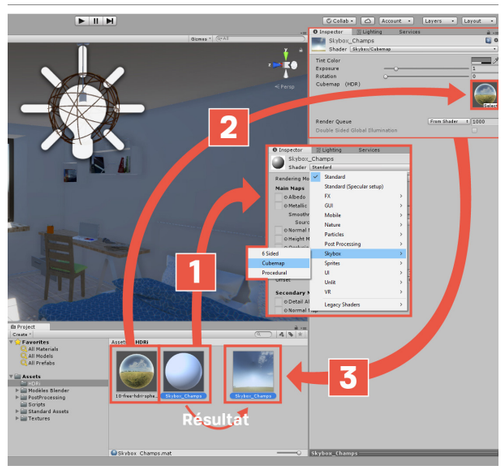
Pour créer une skybox à partir d'une image HDRi sur Unity, il faudra créer ou récupérer une image HDRi qui servira de support à notre nouveau matériau. Après avoir importé l'image dans le dossier ***Assets***, il est nécessaire de changer le type de la texture en cliquant dessus puis en modifiant ces propriétés dans la fenêtre « inspector ». La forme de la texture est à changer, dans l'option ***Texture Shape***, passer de ***2D*** à ***Cube*** et dans les options de qualités au bas de la section, il est possible d'augmenter la résolution utilisée ou non de la texture selon les besoins en qualité (figure 132).

Ceci étant fait, il faut maintenant créer le matériau qui va devenir la skybox de notre environnement. Effectuez un clic droit dans la fenêtre ***Project*** à côté de la texture, et cliquez sur ***Create > Material***. Renommez-le de manière à la retrouver facilement, puis utilisez l'inspector pour changer le type du matériau en skybox, avant de glisser-déposer notre texture. Nous devrions voir apparaître le matériau avec la texture dans l'aperçu de la fenêtre project (figure 133).

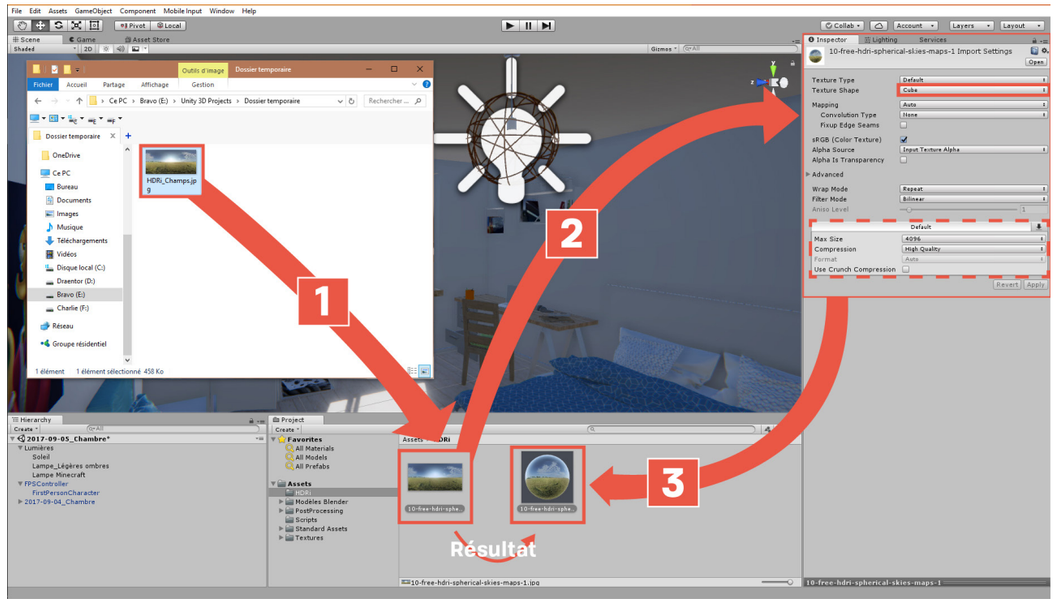
1. Skybox, Unity Documentation, 2017



131. Exemple HDRI - Louvre



132. Unity - Skybox texture



133. Unity - Importer + modifier texture

Il faut maintenant glisser-déposer cette skybox dans la fenêtre adéquate. Les options de lumière se situent dans la barre d'outils en haut de la fenêtre, sous ***Windows > Lightning > Settings***. Cette fenêtre prend en charge tous les paramètres globaux d'éclairage du projet, et permet de modifier l'intensité de la skybox, ainsi que les paramètres graphiques en lien. Nous allons donc glisser-déposer la skybox crée précédemment dans l'option ***Skybox Material*** afin de l'appliquer à notre scène.

Il est encore possible de changer certains paramètres, par exemple, en modifiant l'exposition de notre skybox dans l'inspecteur, et en diminuant la quantité de lumière apportée par la skybox sous l'onglet ***Lightning***. Les paramètres sont à ajuster en fonction de chaque scène (figure 134).

Lights Gameobject

Le rôle de la skybox est de donner un éclairage ambiant, mais il est nécessaire d'utiliser des éclairages annexes pour illuminer correctement la scène.

Unity comprend de base plusieurs types d'éclairages, qu'il est possible de créer depuis la barre de menus, sous ***GameObject > Light***, mais aussi directement depuis la fenêtre ***Hierarchy*** en effectuant un clic droit. Chaque lumière peut être modifiée via les options sous la fenêtre ***Inspector***. Il est possible de désactiver ou non les ombres, afin de profiter de la géométrie pour créer des effets, ou seulement générer une lumière ambiante générale.

FPSController

La caméra est géré par un composant spécifique, permettant de voir la scène et de la parcourir. Ce ***GameObject*** est spécifique pour la vue à la 1ere personne, que nous allons utiliser dans toutes nos scènes.
Pour permettre d'évoluer dans cette scène, il faut placer un ***Gameobject*** dédié. Cet élément n'est pas disponible de base, il nécessité d'être importé depuis les assets standards de Unity, sous le menu ***Assets > Import Packages > Characters*** (figure 135).

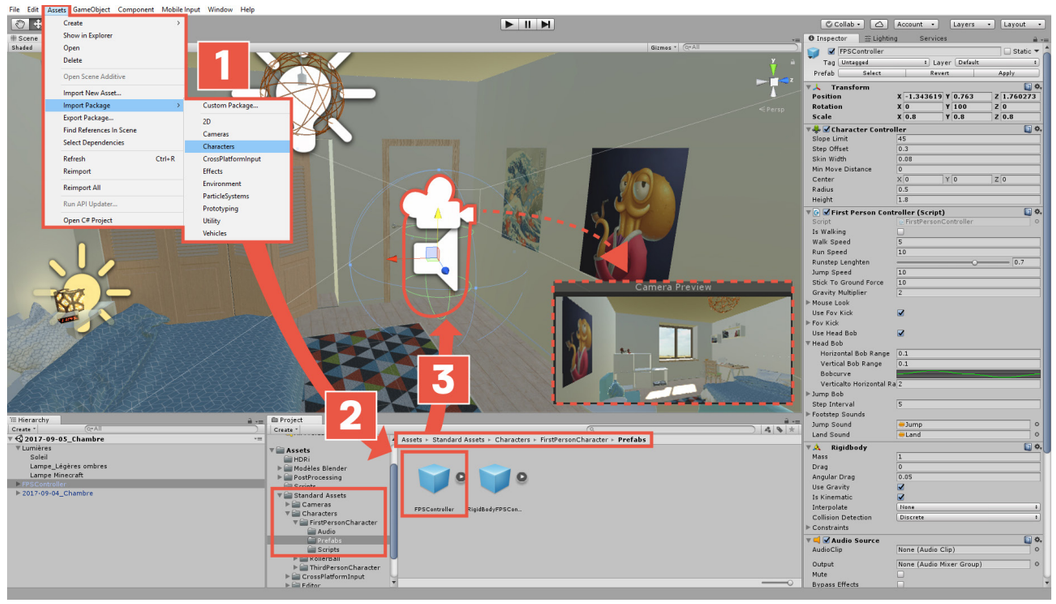
Une fois cette étape effectuée, nous allons pouvoir importer un élément appelé ***Prefab*** abréviation de pré-fabriqué, prenant déjà en charge les fonctions les plus basiques de déplacement dans une scène. Dans notre cas, il s'agira d'importer le prefab ***FPSController*** offrant la possibilité de se déplacer avec une caméra à la première personne. Il est disponible dans la fenêtre ***Project***, sous ***Standard Assets > Characters > FirstPersonCharacter > Prefabs***. Comme précédemment, pour le mettre en place, il suffit de le glisser-déposer dans la fenêtre ***Scene***. Comme tout objet, il est possible de le redimensionner via l'outil ***scale***, mais aussi de le déplacer et le tourner afin d'avoir la caméra dans la direction voulue.

Dès lors, sous l'onglet ***Game***, il est possible d'avoir un aperçu de notre caméra, et de commencer à parcourir notre scène en cliquant sur le bouton ***Play*** au-dessus de celle-ci.

Les déplacements se font avec les touches du clavier, et quitter la démonstration s'effectue en appuyant sur ***Echap*** puis en cliquant à nouveau sur le bouton ***Play***.

Afin de conforter l'aspect visuel, et gérer plus en profondeur ses paramètres, il est possible de créer un profil de gestion du post-processing, c'est à dire une gestion de tous les effets en temps réel d'un moteur de jeu. Dans la fenêtre ***Hierarchy***, il est possible de dérouler le prefab ***FPSController*** ajouté auparavant pour découvrir la caméra à proprement parler, l'objet ***FirstPersonCharacter***. Cet objet a un rôle important dans la scène car il permet de définir tout ce que le joueur observera dans son environnement.

134. Unity - Rendu en jeu



135. Unity - Assets location

Post-Processing

Le post-processing est l'application des options de corrections et effets évoqués auparavant. Cela va nous permettre de corriger les défauts visuels, tels que l'aliasing ou le filtrage des textures éloignées, ainsi que créer des effets supplémentaires attachés à la caméra que nous venons de placer dans la scène, tels que le ***bloom***, ou la correction de couleurs.

Chaque objet possède une série de propriété visibles dans la fenêtre ***Inspector*** et nous allons pouvoir interagir avec ces propriétés, afin d'en créer de nouvelles permettant de générer de nouvelles fonctions dans notre scène. Pour gérer le post-processing de notre caméra, cliquez sur ***Add Component*** et recherchez ***Post Processing Behaviour*** un script relié à un profil que nous allons créer.

Une fois cette étape effectuée, dans la fenêtre ***Project***, effectuez un clic droit et créer un nouveau ***Post Processing profile***, qui contiendra toutes les options décidées par l'utilisateur. Ces options ont déjà été présentés auparavant, nous allons alors relier notre profil à la caméra, en sélectionnant à nouveau la caméra, c'est à dire l'objet ***FirstPersonCharacter*** dans la fenêtre ***Hierarchy*** puis en effectuant en glisser-déposer de notre profil nouvellement crée à l'option ***Profile*** du component ajouté auparavant à notre caméra (figure 136).

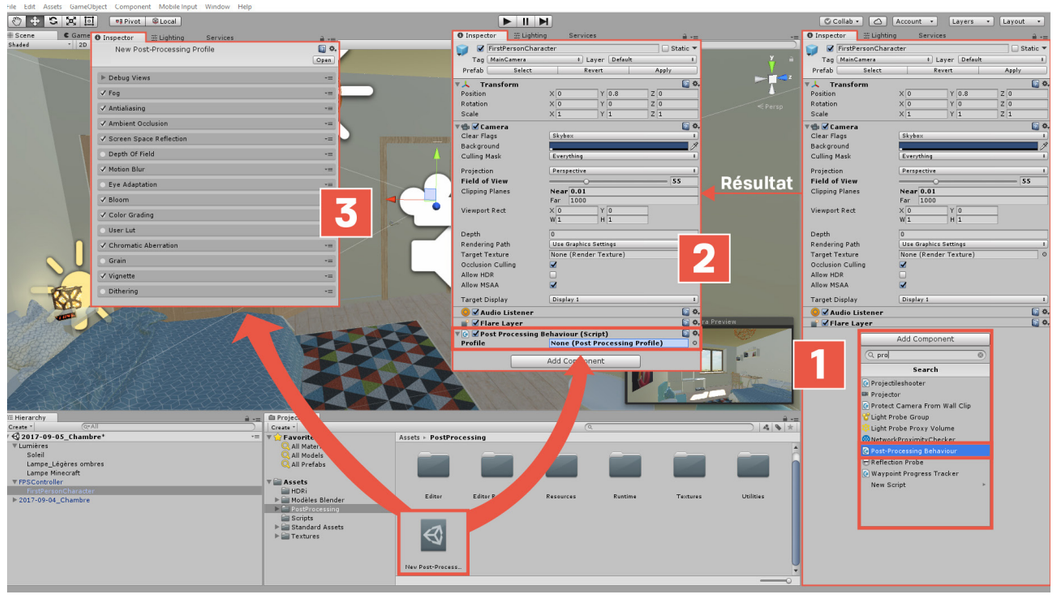
Les options de post-processing sont toujours disponibles en cliquant sur le script crée dans la fenêtre ***Project***, chaque scène nécessite un réglage particulier, selon les nécessités de la visualisation.

S'adapter au public

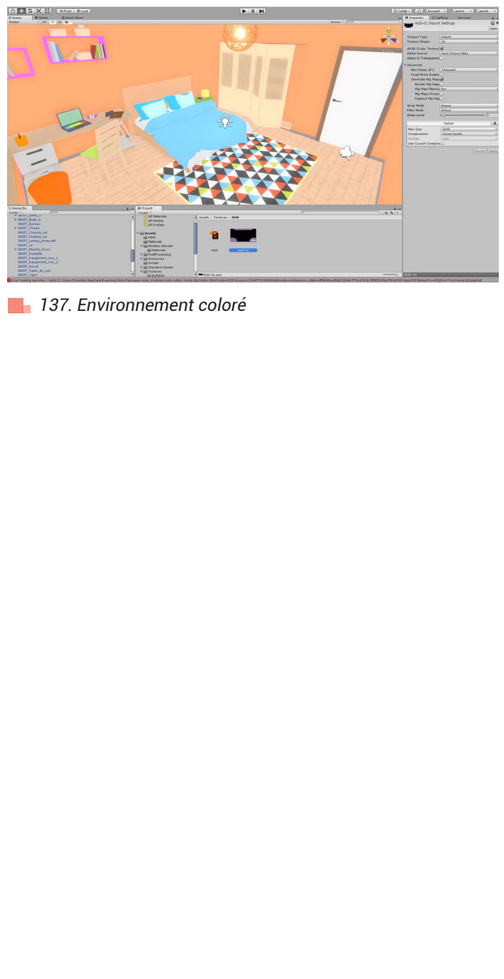
Les deux autres scènes ont pour objectif de créer une autre approche de la même scène, en modifiant quelques paramètres, afin de créer un tout autre rapport en n'effectuant que quelques modifications depuis le moteur Unity. L'objectif est alors de créer trois univers différents basés sur une seule et unique scène, afin de mettre en avant la capacité du moteur à permettre une multiplicité de représentation.

La seconde scène sera spécifique par le traitement effectuée sur les textures et les ***shader*** (figure 137). En effet, l'objectif est de s'adresser à des enfants, en les mettant dans la peau d'un jouet, parcourant la scène et découvrant un nouveau rapport d'échelle.
Le travail sur l'ambiance se fera notamment par la colorimétrie, en utilisant une palette de couleurs vives et chaudes. De plus, l'affichage des couleurs se fera via l'utilisation d'une fonction spécifique d'un ***shader***, permettant d'effacer les ombres et de permettre une lecture plus claire des couleurs pour l'enfant.

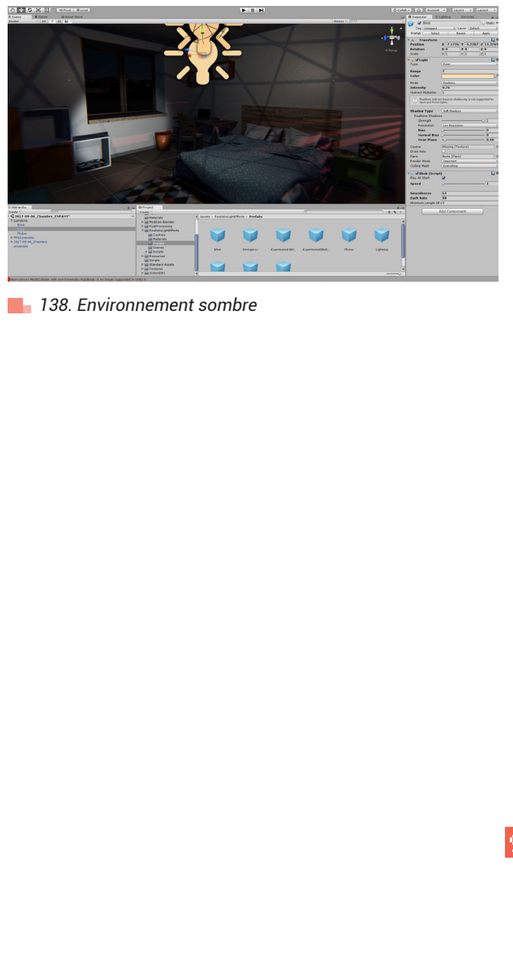
La dernière scène reprendra le travail effectué sur les textures pour la scène réaliste, mais prendra par dans un environnement beaucoup plus sombre, dans lequel nous nous placerons à l'échelle d'un enfant. Cette scène cherche à s'adresser à un public de jeu d'horreur en particulier, en mettant en avant le travail de l'éclairage et des ombres au travers des objets, en nous plaçant dans un environnement avec pour seul éclairage une lampe torche (figure 138). L'objectif est de retranscrire une atmosphère effrayante, en se basant uniquement sur le jeu d'ombres et de lumière.



136. Unity - Glisser déposer script post-process



137. Environnement coloré



138. Environnement sombre

Dans la tête d'un robot

Afin de générer cette ambiance colorée destinée aux enfants, nous allons utiliser des fonctions inhérentes au ***shader***, modifier l'éclairage ambiant pour qu'il n'interagisse plus avec notre scène et réduire la dimension de notre caméra pour l'adapter à notre point de vue.

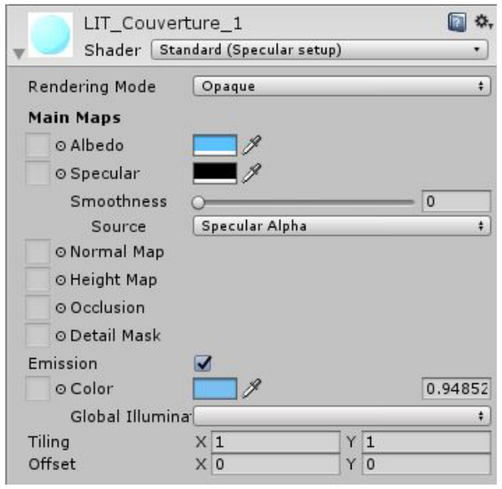
Chacun des ***shaders*** utilisés pour créer notre scène de base comprend une option ***Emission***, déjà utilisée pour certains objets, dont le réveil, permettant d'afficher l'heure digitale et de signifier sa luminosité. Comme évoquée précédemment, activer l'option ***emission*** d'un objet lui ôte toute capacité de répondre à la lumière par des effets, puisque l'objet en émet-lui même. Cela nous permettra alors de créer une ambiance colorée, sans aspérité, car la lumière ne peut plus interagir avec la texture. Pour réaliser cela, cliquez sur l'objet et paramétrer son ***shader*** en lui ôtant toute texture, en diminuant l'option ***smoothness*** au minimum afin d'éviter toute forme de reflet en définissant le matériau comme rugueux. Enfin, activer l'option ***emission*** et définissez une couleur proche de la couleur d'origine de l'objet (figure 139).

Afin d'éviter toute perturbation de l'éclairage ambiant sur la scène, ouvrez la fenêtre ***Lightning*** sous ***Window > Lightning > Settings***, et enlever la skybox précédemment instaurée. Vous pouvez aussi supprimer toute les lumières crées auparavant.

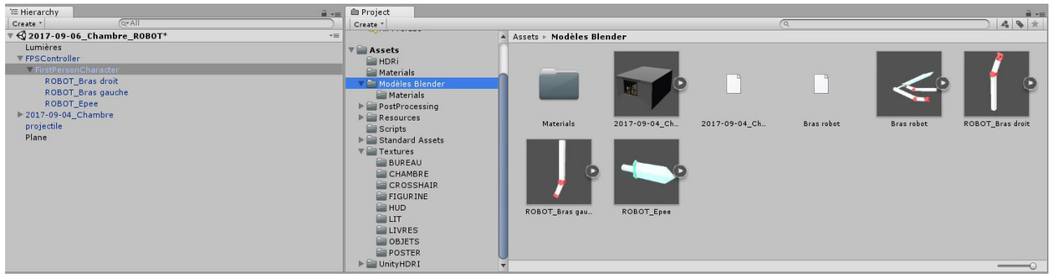
Ensuite, il est aussi nécessaire de réduire la taille de notre ***GameObject FPSController*** afin de l'adapter au point de vue recherché. Il faut alors cliquer dessus, et modifier les dimensions en X,Y et Z via l'option ***Scale*** de l'***Inspector***. Vérifiez toutefois que le moteur permet de continuer à parcourir la scène après modification. Dans le cas contraire, agrandissez la scène plutôt que ce ***GameObject*** pour éviter toute erreur.

La dernière étape consiste à modifier quelques textures afin de les adapter au style graphique de la scène, notamment le plancher et le tapis, dont il faut effacer le relief afin de le rendre parfaitement lisse.

Enfin, pour immerger un peu plus l'utilisateur dans cette scène, nous pouvons modéliser deux bras d'un jouet robot ainsi qu'une épée, afin de intégrer à la caméra. Pour cela, après avoir réalisé vos modèles et les avoir importé dans le dossier ***Assets***, glissez-déposer les sous le ***GameObject FirstPersonCharacter*** du ***FPSController***, afin qu'ils soient intégrés à l'objet et bougent de manière identique. Vous pouvez vous aider de la vision de la caméra pour les placer correctement et donner l'impression de voir au travers de la tête d'un robot (figures 140 & 141).



139. Modification du shader



140. Glisser-Déposer les modèles dans la hierarchy

141. Visualisation des bras du robot

SAW6

A nouveau, nous repartons de la scène réaliste. Nous allons chercher à modifier la lumière afin de créer une nouvelle ambiance puis nous allons adapter notre caméra au point de vue d'un enfant pour donner une présence physique tout en gardant le sentiment d'être plus petit parmi des objets plus volumineux.

La première étape est d'utiliser une nouvelle skybox à partir d'un nouveau HDRi de nuit, pour simuler au mieux l'éclairage de nuit de la scène. Comme précédemment, après avoir sélectionné votre HDRi, et avoir crée un nouveau matériau pour l'accueillir, glisser-déposer-le sous l'option ***Skybox*** de la fenêtre ***Lightning***.

Vous pouvez aussi supprimer toutes les sources de lumières, car nous allons en créer d'autres, à partir d'un ***plugin*** disponible sur l'***asset store***. Pour accéder l'***asset store***, cliquez sur ***Window > Asset store*** depuis la barre d'outils, et recherchez le ***plugin Realistic Light Effects / Animations***, qui génère des scripts pour différents effets de lumières à partir des objets de base de Unity (figure 142).

Vous avez normalement accès à des nouveaux éléments dans la fenêtre ***Project***, dont un dossier ***RealisticLightEffects > Prefabs*** contenant les lumières que nous allons utiliser. Dans notre cas, nous allons glisser-déposer des lumières ***Blink*** et ***Flicker*** dans notre scène, pour créer des ambiances lumineuses différentes et dynamiques. Le choix des emplacements des différentes lumières peut être différent, le but est surtout de permettre un jeu par les ombres. Chacune de ces lumières contient un ***component*** permettant d'ajuster le grésillement ou le clignotement.

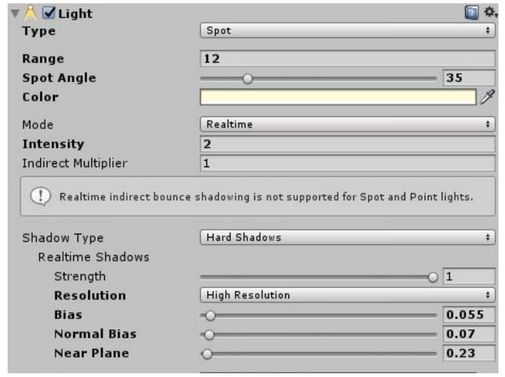
Enfin, comme précédemment, nous allons créer un modèle 3D de lampe torche pour l'attacher à notre caméra et la faire suivre le mouvement effectué par le joueur. En plus, nous allons y rajouter une lumière, placée devant la lampe torche pour simuler son effet sur l'environnement. Pour cela, il est nécessaire de passer le type de la lumière en ***Spot*** et de régler les options suivantes selon l'échelle de la scène et les contraintes (figure 143). De cette manière, les lumières de la scène agiront dynamiquement, et le joueur sera capable d'éclairer la partie de l'environnement qu'il souhaite (figure 144).

Liberté de coder

La grande liberté d'utilisation de Unity est aussi une faiblesse pour les amateurs, à qui cela nécessite une longue période d'investissement avant de pouvoir en tirer tous les avantages. De plus, le champ des possibilités est vaste, car la majorité des éléments ne sont pas codés, ce qui crée une autre difficulté dans la personnalisation d'un projet au complet.

Générer une visualisation simple et efficace d'un projet est un objectif facilement atteignable avec les moyens existants et l'aide de la communauté autour du moteur. Toutefois, les fonctions avancées ne sont pas évidentes à développer et posent encore des problèmes pour un moteur en constante évolution. Plus un moteur est général, moins il sera efficace dans un domaine précis.

142. Plugin de l'asset store



143. Modification de la lampe

144. Eclairage par la lampe torche

Présentation

On appelle réalité virtuelle, une technologie permettant de simuler la présence d'un utilisateur dans un environnement virtuel par l'utilisation d'un casque ou de lunettes. L'attrait pour la VR (Virtual reality ou Réalité virtuelle) s'est beaucoup développé ces dernières années avec l'apparition de dispositifs grands publics donnant accès plus facilement à cette technologie.

Nous pouvons citer la société Oculus VR, et son dispositif l'Oculus Rift, dont le premier prototype date de 2011, mais aussi le casque HTC Vive, développé en collaboration par HTC et Valve, développeur de jeux vidéos (figure 145).

Le lien entre le jeu vidéo et la réalité virtuelle s'est fait très naturellement, puisque les deux univers nous immergent dans un environnement virtuel, la seule différence étant le médium. Toutefois, la réalité virtuelle ne s'appréhende pas de la même manière qu'une interface de visualisation classique, car les sensations et l'appréhension du monde sont beaucoup plus présentes à l'aide d'un casque VR. C'est pourquoi cette technologie et ses possibilités nous intéressent, et nous allons chercher à voir en quoi elles vont nous permettre de ressentir l'espace et faire découvrir un projet ou un lieu plus intensément qu'avec une démonstration sur écran.

Solutions existantes pour l'architecture

Il existe actuellement des solutions clés en main pour découvrir un projet par la réalité virtuelle depuis des outils de type BIM. Revit comprends maintenant Autodesk Live, permettant de créer un fichier parcourable avec un casque VR et d'en changer quelques paramètres, tels que l'ensoleillement l'angle de la lumière (figure 146).

Une autre solution est le logiciel Enscape, conçu pour fonctionner avec Revit et reprends beaucoup de fonctions évoquées auparavant, en permettant de se déplacer dans l'environnement par téléportations, tout en changeant quelques paramètres inhérents à la lumière (figure 147).

Ces outils se destinent aux architectes afin de créer un modèle expérimental pendant la phase de conception,et ne destinent pas à une visualisation d'un projet terminé.1

Pour développer une visualisation complète, il faudrait se tourner vers des outils plus complets, tels que Autodesk Stingray, qui est un moteur de jeu développé par Autodesk, capable de prendre en charge des fichiers issus des différents logiciels de leur suite (figure 148).

L'enthousiasme des architectes pour ces solutions vient surtout de la prise de conscience du volume par la réalité virtuelle, et les possibilités de développement et de conception qu'elles pourraient offrir. Cela pose la question de la capacité de l'architecte à concevoir, sans ses outils, et nous fait nous demander si un tel support technologique aurait vraiment un intérêt pour la conception. On retrouve des problématiques similaires à la question de l'impression 3D et de la production de maquette, où la perception du volume est essentielle.

De notre côté, nous allons rechercher en quoi l'apport de la réalité virtuelle présente un intérêt pour une visualisation avancée, à partir des outils utilisés précédemment, et permettant de faire évoluer le spectateur dans des scénarios différents à partir d'un même outil de base.

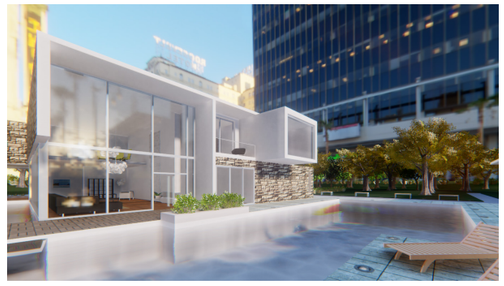
Fonctionnement

Le casque HTC Vive, développé par HTC et Valve est pleinement et très facilement compatible avec le moteur, de par l'ajout d'un plugin officiel de Valve sur l'Asset Store. Dès

1. From Revit to VR, Février 2017, AECMagazine

145. HTC Vice / Oculus Rift

146. Autodesk Live



147. Enscape

148. Autodesk Stingray

lors il nous sera très facile de l'utiliser.

L'adaptation d'une scène à la réalité virtuelle nécessite des changements, spécifiquement vis-à-vis de la caméra et de la manière de se déplacer. En effet, l'utilisateur peut maintenant regarder ce qu'il veut, sans utiliser de support autre que son corps. Les interactions sont alors différentes, car notre cerveau est incapable de gérer correctement nos déplacements dans un univers virtuel alors que nous ne bougeons pas dans la réalité. Le casque HTC vive permet une petite liberté de déplacement grâce à l'installation de capteurs autour d'une petite zone d'environ 2,5x2,5m, ce qui peut être très peu à l'échelle d'un environnement complet (figures 149 & 150).

C'est pourquoi les développeurs ont créé un système de déplacement basé sur la téléportation à l'aide de manettes fournies avec le casque.

S'immerger dans l'environnement

La capacité de la réalité virtuelle à nous amener dans un monde est extrêmement grande, en plus d'avoir la capacité de nous faire ressentir les dimensions de manière très précises. La plupart des jeux vidéos exploitant la VR sont d'ailleurs très souvent des jeux à la première personne, tels que ***SUPERHOT VR***, ***Resident Evil VII***, exploitant cette technologie pour nous faire ressentir l'environnement et ce que le personnage vit au travers d'un scénario d'une manière plus intense (figures 151 & 152).

Certains jeux de courses, comme ***Dirt Rally*** ou ***Project CARS*** utilisent aussi la VR, de manière à nous faire ressentir la conduite par le casque (figure 153).

Pourtant, cette perception de la profondeur est difficilement exploitée pour d'autres usages et il existe peu de cas de jeux VR qui se jouent à la troisième personne. Il existe tout de même quelques exemples, comme ***Lucky's Tale***, un jeu de plateforme permettant de suivre le héros du jeu au travers d'environnements et mettant à profit la VR pour mieux appréhender l'environnement (figure 154).

La capacité de la réalité virtuelle à nous amener dans un monde est extrêmement grande, en plus d'avoir la capacité de nous faire interagir avec de façon plus poussée, de par l'utilisation des manettes servant de remplaçant de nos mains dans le monde virtuel. Cette recherche d'une atmosphère et d'une ambiance est alors encore plus efficace par le biais de la VR.

Adapter sa représentation

L'intérêt est alors d'utiliser le casque de réalité virtuelle comme nouveau support d'interaction pour nos différentes scènes, afin d'accentuer le rapport avec le volume crée. De plus, la gestion des contrôleurs ouvre un tout autre aspect de l'interactivité, en simulant nos mains et créant un nouveau palier d'appropriation de l'espace virtuel.

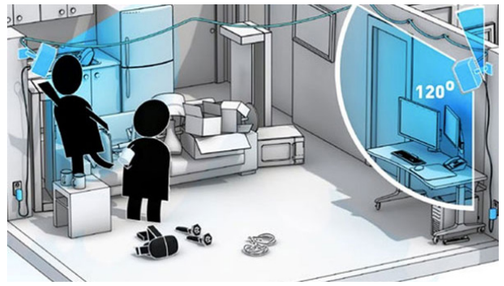
Il s'agit alors de mettre en pratique ces outils grâce au moteur de jeu Unity et d'améliorer nos différentes scènes afin de créer une nouvelle manière de découvrir et de parcourir les différents scénarios.

Pour cela, nous proposons un protocole d'utilisation étant donné que l'expérimentation n'a pas pu être réalisée du à des contraintes extérieures imprévues. Ainsi, la majorité de ces explications seront alors théoriques, bien que fonctionnelles et ayant été testées pour certaines plus tôt durant l'année.

Nous allons commencer par voir comment adapter les outils du casque à la scène afin de pouvoir se déplacer, puis comment gérer les différents objets de la scène afin de fournir une interaction basique avec quelques éléments.

HTC Vive et Unity 5

Tout d'abord, la transition entre le moteur de rendu et le casque de réalité virtuelle nécessite des adaptations. Pour cela, nous nous appuyons sur la scène réalisée précédemment.



149. Positionnement des capteurs

150. Zone d'évolution

151. SuperHOT VR

152. Resident Evil VII

153. Project Cars

154. Lucky's Tale

Il est nécessaire de supprimer la caméra présente dans le projet Unity afin de mettre en place un nouveau prefab fourni avec le plugin SteamVR spécialement conçu par Valve pour le moteur Unity.

Le casque nécessite une installation complète disponible depuis le logiciel Steam, et très bien expliquées dans le mode d'emploi. Avant de pouvoir utiliser le casque, n'oubliez pas de démarrer Steam et l'outil SteamVR.

Depuis l'asset store, récupérez le ***SteamVR Plugin*** qui comprend tous les outils nécessaires au fonctionnement du casque sur le moteur.

Pour pouvoir se déplacer dans l'espace, il va falloir placer une caméra disponible dans le plugin, et utiliser des zones au sol définissant les points et espaces dans lesquels le joueur peut se déplacer et se téléporter.

La première étape est de glisser-déposer dans la scène ou la hiérarchie le prefab ***CameraRig*** disponible dans ***Assets > SteamVR > Prefab***. Cet objet contrôle la position du casque et des manettes dans l'environnement. Pour être sur du bon fonctionnement, n'oubliez pas de supprimer l'objet ***Main Camera*** s'il existe ou tout autre caméra pouvant interférer[2].

Se déplacer

Se déplacer dans un environnement en VR pourrait se faire de manière classique, à l'aide d'une manette, d'un clavier et l'appui sur des touches créant un déplacement dans une direction voulue. Mais nous avons vu précédemment que cette démarche est difficile à utiliser, car elle met en contradiction notre corps immobile, et le monde virtuel dans lequel nous bougeons. Dès lors, des méthodes de déplacement alternatives se sont développés, notamment grâce à l'utilisation des contrôleurs pour se téléporter sur des points ou des zones prédéfinies par le développeur.

Dans la fenêtre ***Hierarchy***, développez l'objet ***CameraRig*** de manière à voir les trois objets qui y sont rattachés, c'est à dire les deux contrôleurs et le casque. Dans ce même groupe, créer deux ***Empty Children***, à renommer ***Right Hand*** et ***Left Hand*** (figure 155).

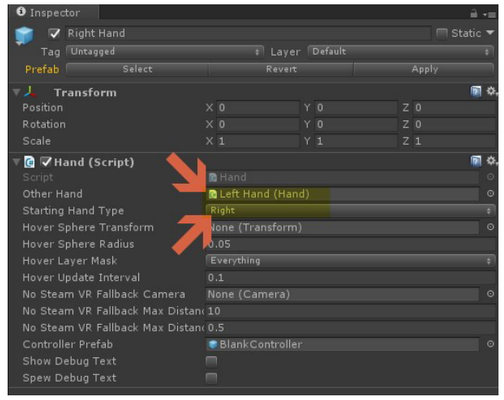
A chacun de ces objets, il est nécessaire d'ajouter le component ***Hand*** depuis l'inspecteur. Toujours dans l'inspecteur, glissez-déposer sous l'option ***Other Hand***, l'autre main. C'est à dire, glisser-déposer l'objet crée ***Left Hand*** dans l'option de l'objet ***Right Hand*** et inversement. De plus, dans l'objet concernant la main droite, sous l'option ***Starting Hand Type*** sélectionnez ***Right*** (figure 156).

Pour utiliser la téléportation, cliquez sur l'objet ***CameraRig*** et ajoutez-lui le component ***Player***. Dans ce component, sous l'option ***Hmd Transforms »***, glissez l'objet ***Camera (eye*** de l'objet ***CameraRig*** et plus bas, augmentez l'option ***Size*** à deux pour pouvoir y glisser-déposer les deux objets concernant les mains crées précédemment (figure 157).

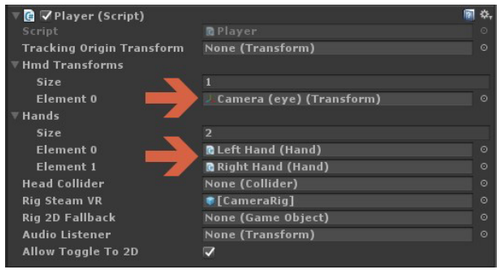
Dans le dossier ***Assets > SteamVR > InteractionSystem > Teleport > Prefabs*** vous trouverez deux prefabs, ***Teleporting*** et ***TeleportingPoint***. Le premier est l'objet nécessaire au bon fonctionnement de cette fonction tandis que le second est un GameObject permettant de définir un point dans lequel le joueur peut se téléporter grâce aux contrôleurs. Vous pouvez en placer quelques-uns sur la scène afin de commencer à éprouver le système en lançant la démo (figure 158).

Pour créer une zone de téléportation, créer un objet plan au sol, et renommez-le comme vous le souhaitez. Déplacez-le de manière à ce qu'il soit très légèrement au-dessus de la scène, assez pour que les textures des deux éléments ne se mélangent pas tout en faisant en sorte qu'il ne donne pas l'impression de flotter (figure 159).
Cliquez sur l'objet et ajoutez-lui le component ***Teleport Area*** après avoir décidé de la taille nécessaire à l'objet. Le matériau est normalement devenu transparent, pour n'afficher que la grille lorsque vous la pointerez à l'aide du contrôleur[3] (figure 160).

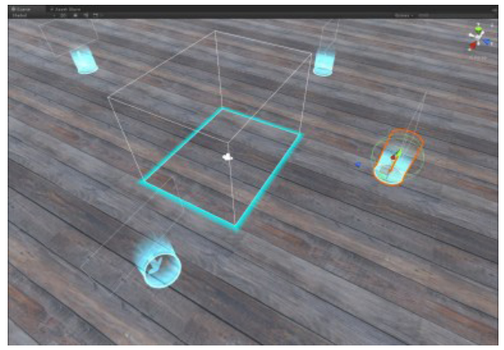
155. Créer empty



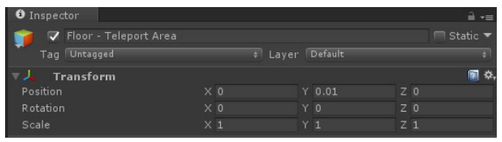
156. Correspondance des options



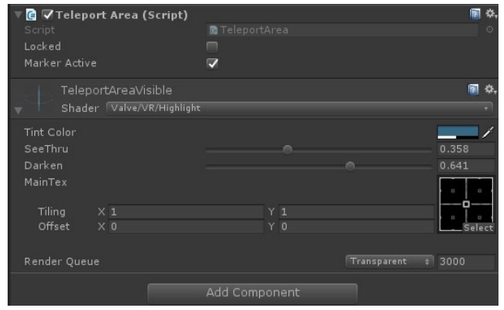
157. Player component



158. Teleport points



159. Coordonnées du plan



160. Grille au sol

2. HTC Vive tutorial for Unity, 22 décembre 2016, Eric van de Kerckhove

3. SteamVR Locomotion and Teleportation Movement, 16 Mai 2017, Jason Weimann

Interagir

L'interaction grâce aux contrôleurs du HTC Vive permet d'envisager une nouvelle manière de voir et comprendre l'espace. Pour générer cette interaction il s'agira de préparer les composants de manière à ce qu'il soient utilisables et reconnaissables en tant que tel par le moteur.

Nous allons pouvoir réutiliser les composants établis précédemment, en complétant leurs fonctions.

Sélectionnez l'objet ***Left Hand*** crée à l'étape précédente, et sous l'option ***Controller Prefab***, assignez-lui le prefab ***BlankController***. Cela peut être fait en glissant-déposant depuis le dossier ***SteamVR > InteractionSystem > Core > Prefabs***, ou en recherchant le prefab après avoir cliqué sur le petit cercle à droite de l'option. Répétez cette opération pour l'objet ***Right Hand*** (figures 161 & 162).

Ensuite, sélectionnez l'objet ***Camera (eye)*** sous ***CameraRig*** et à l'aide du clic droit, créer un nouvel objet 3D ***Sphere***. Dans l'inspecteur, changez les paramètres de la sphère, en diminuant son volume, puis en désactivant le rendu de la sphère à l'écran, en cliquant sur l'option ***Mesh Renderer***. L'objectif ici, est d'utiliser la sphère comme point de contact invisible avec tout les objets dont nous voudrons une interaction (figures 163 & 164). Sélectionnez l'objet Player, et glissez-déposer l'objet sphere nouvellement crée sous l'option ***Head Collider*** du component Player, puis juste dessous, glissez-déposer tout le gameobject CameraRig dans l'option ***Rig Steam VR*** (figures 165 & 166).

Enfin, créer un objet 3D simple, cube, sphere, cylindre ou sélectionnez un objet indépendant importé dans votre scène, et ajoutez-lui le component ***Throwable*** (figure 167).

Vous pouvez alors essayer la scène et attraper puis lancer tout les objets comportant ce component grâce à la gâchette de votre contrôleur.[4]

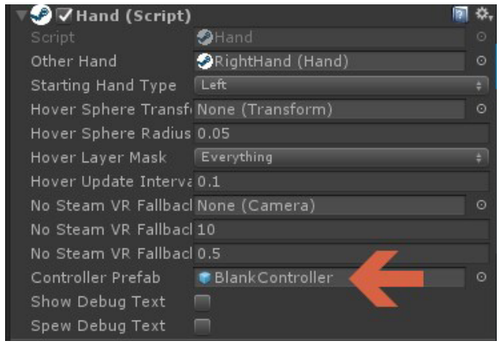
Vivre les scénarios

Cette nouvelle dimension d'interaction permet une approche plus avancée du projet et une contextualisation beaucoup plus forte en intégrant le spectateur au coeur de la scène.

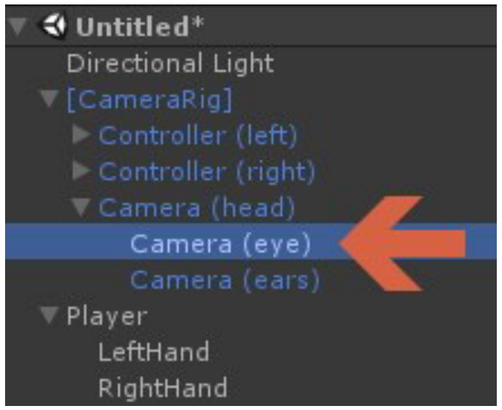
Le jeu vidéo remplit déjà cet objectif depuis sa création et l'utilisation de ces outils pour l'architecture permet d'ouvrir une nouvelle forme de communication avec le public. Nous avons déjà testé cet aspect lors des journées portes ouvertes de l'Ecole d'architecture de Nancy, où une démonstration de l'utilisation du casque a été mise en place, pour permettre aux personnes de découvrir l'environnement virtuel recrée pour l'occasion. L'enthousiasme lors de la prise de conscience de l'échelle et du volume par le casque est déjà un premier pas, qui peut être amélioré par les outils évoqués auparavant.

Le potentiel des contrôleurs pourrait par exemple être utilisé pour réaménager l'ensemble de la première scène réaliste. En préparant les différents modèles de manière à ce qu'ils soient interactifs, nous pourrions imaginer une appropriation de l'espace avant même sa réalisation. Nous pourrions aussi utiliser les capacités du moteur pour générer les différents éclairages et simuler plusieurs heures, climats d'une même journée, le tout en temps réel.

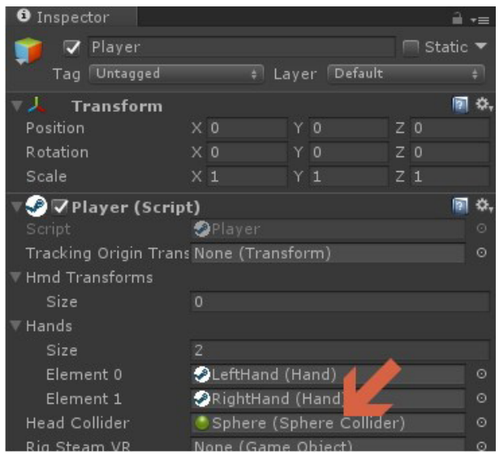
La seconde scène pourrait permettre de développer un jeu à part entière, avec une interaction par l'épée, un jeu de combat pour enfant contre des robots. Tandis que la dernière scène pourrait générer une immersion beaucoup plus poussée, par le contrôle direct du joueur de sa lampe torche, avec la possibilité de regarder et éclairer ce qu'il souhaite, tout en attrapant des objets, se dessinant un parcours le long d'un jeu d'énigme amenant à découvrir l'architecture d'une toute autre manière.



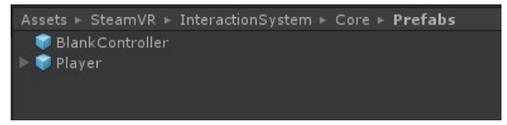
161. Assigner le Blank Controller



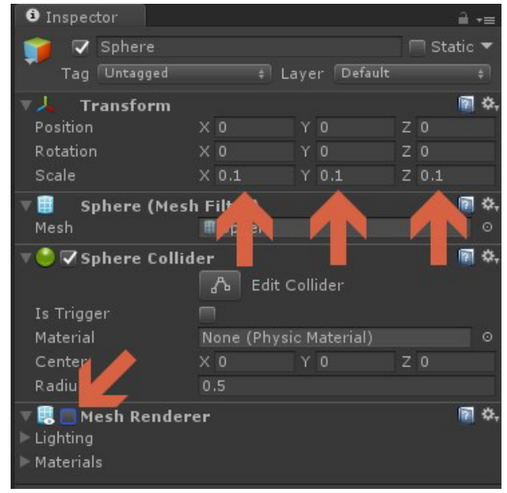
163. Sélectionnez Camera (eye)



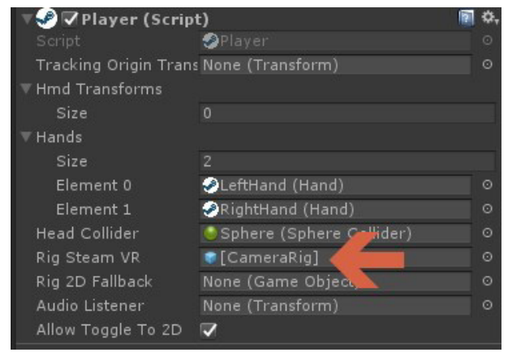
165. Glisser-déposer la sphère sous Head Collider



162. Emplacement du prefab



164. Désactiver le Mesh Renderer



166. Glisser-déposer CameraRig sous Rig Steam VR

167. Ajouter le component Throwable

4. VR Interaction – SteamVR Lab Interaction System – Throwables, 19 Mai 2017, Jason Weimann

FICHE RAPPEL
DE LA MODÉLISATION À LA RÉALITÉ VIRTUELLE

BLENDER

+

- Grande diversité d'outils
- Beaucoup de possibilités par le système de nodes
- Grande communauté
- Bonne capacité d'importation et d'exportation
- Possibilité de bake
- Gère l'UV Mapping
- Gratuit
- Open Source

–

- Interface lourde
- Fonctions parfois difficile à trouver
- Moteur de visualisation daté
- Impossibilité de gérer des calques de peinture
- Système de calques contre-intuitif

UNITY 5

- Beaucoup de fonctions
- Création de plusieurs types de jeu
- Grande communauté
- Asset Store
- Multi-plateformes
- PBR intégré
- Support de la VR

- Code source verrouillé
- Pour des options avancés, il faut maîtriser le code
- Code de programmation maison
- Trop peu spécialisé sur certains points

HTC VIVE

- Très bonne expérience
- Suivi à l'échelle d'une petite pièce
- Interface
- Contrôleurs

- Prix
- Lourd

MODÉLISATION

- Modéliser avec le moins de *vertices* possible
- Limiter les polygones
- Penser la texture au préalable
- Vérifier le sens des normales
- Fermer les volumes

UV MAPPING

- Réfléchir au dépliage du modèle
- Si faces identiques, les superposer

TEXTURING

- Limiter les tailles de textures
- Utiliser différentes couches pour créer du détail
- Baker plusieurs textures de petits objets en une seule texture pour économiser de la place
- Le minimum de textures utilisées, le minimum de ressources utilisées

CAMERA

- Penser à la position de la caméra
- Effets de post-processing

SHADERS

- Utiliser les bonnes textures au bon emplacement
- Différence entre *Metallic* et *Dielectic*

PROJECT

- Ranger les fichiers
- Renommez les fichiers correctement

FONCTIONNEMENT

- Utiliser le plugin officiel
- Penser à lancer SteamVR avant Unity

CONCLUSION

Au fil de la découverte des règles de la photographie et des outils du jeu vidéo, nous avons mis en évidence la capacité de cette industrie à s'adapter aux changements et à la diversité des intentions graphiques développées par chaque jeu.
En effet, la question du sens développé par l'image s'accorde avec les nouveaux outils numériques permettant une visualisation architecturale. Plusieurs règles et méthodes se sont établies, permettant de construire un modèle, lui conférer une texture et gérer l'utilisation d'un moteur afin d'établir une scène complète, déclinée en plusieurs ambiances.

Un moteur de jeu a la capacité d'absorber les contraintes de l'images, du rendu et de la visualisation architecturale, afin de fournir un projet qu'il est possible de représenter de différentes manières, en détournant ces règles. Dès lors l'ensemble des techniques expliquées ont un rôle à jouer dans la construction de l'environnement et son parti pris graphique.

Construire une visualisation du projet en l'abordant par la photographie permet de commencer à construire un sens à l'image et de donner un rôle ainsi qu'un contexte au projet. Toutes ces notions peuvent ensuite être adaptées à la technique, afin de réaliser le modèle architectural amenant à un modèle 3D complet, offrant énormément de possibilités. Ainsi, l'utilisation du PBR offre une approche de la texture qui fonctionne avec la lumière et perme de varier les contraintes extérieures au modèle pour créer diverses ambiances à l'attention de plusieurs publics.

L'ajout de la réalité virtuelle au processus de visualisation en temps réel permet de créer un nouveau palier dans l'interaction. En effet, nous avons vu que l'utilisation du casque immerge totalement le spectateur dans l'environnement, tandis que les contrôleurs créent un nouveau rapport à la scène. Toutefois, nous pourrions nous demander si l'interactivité présente un intérêt pour le grand public, au delà d'une simple action avec l'environnement. En effet, le rendu qu'il soit réaliste ou non, parcourable ou non peut être consideré comme un objet commercial destiné à la vente d'un produit qu'est l'architecture ou le projet. Le grand public est avant tout destinataire dans une logique d'achat d'un produit fini. Ainsi, nous pouvons nous questionner sur la limite entre l'objet de vente qu'est le rendu en temps réel et l'outil de conception qu'il peut devenir.

BIBLIOGRAPHIE

Article : Boston Culture Center Daytime Perspective
https://visualizingarchitecture.com/project/boston-culture-center-daytime-perspective/

Article : Giotto, Rivage de boheme
http://www.rivagedeboheme.fr/pages/arts/peinture-jusqu-au-14e-siecle/giotto.html

Ressource : Bernd et Hilla Becher, archives du centre pompidou
20 octobre 2004 - 3 janvier 2005
https://www.centrepompidou.fr/cpv/resource/c887Rx5/rBKA9rp

Article : Architectural photography was «staged and contrived» before digital, say Hufton and Crow
Amy Frearson | 12 April 2015
https://www.dezeen.com/2015/04/12/nick-hufton-allan-crow-interview-architectural-photography-digital/

Article : Julius Shulman, Photographer of Modernist California Architecture, Dies at 98
Andy Grundberg, 17 Juillet 2009
http://www.nytimes.com/2009/07/17/arts/design/17shulman.html

Article : The Art of Rendering: How to Create "Hotel 114" Using Cinema 4D and Photoshop
Ronen Bekerman, 29 Mars 2017
https://architizer.com/blog/the-art-of-rendering-how-to-create-hotel-114/

Conférence : L'art de reprendre la main, François Robin
17 Mai 2016, ENSANancy

Are 3D Renderings Deceiving Architects and Clients?
John Kutyla, 6 Octobre 2015
http://www.archdaily.com/774853/are-3d-renderings-deceiving-architects-and-clients

Article : Joseph kosinski talks to archinect about his transition from architecture to hollywood
Paul Petrunia, 5 Septembre 2014
http://archinect.com/features/article/108112212/cutting-room-joseph-kosinski-talks-to-archinect-about-his-transition-from-architecture-to-hollywood

Article : Dishonored 2 ou l'Art au service du jeu
Olivier Pallaruelo, 6 Novembre 2016
http://www.allocine.fr/film/court-metrage/news-18657240/?utm_sourc

Vidéo : Game Architect
Arte, 6 Janvier 2016
https://www.youtube.com/watch?v=wH8X5m7FZ0g

Article : How to Render Your Building to Sell it, Not Just Show it
John Kutyla, 12 Août 2015
http://www.archdaily.com/771736/the-rendering-view-how-to-render-your-building-to-sell-it-not-just-show-it

Vidéo : LALALAND, Ce que disent les couleurs
Le Fossoyeur de films, 30 Janvier 2017
https://www.youtube.com/watch?v=k8Zk4eUSgF4

Article : Les rendus sont-ils mauvais pour l'architecture
Nicolas Richelet, 6 Avril 2015
http://rendus-et-perspectives.com/rendus-et-architecture/

Article : Photographic approach in architectural visualisation
Lasse Rode, 26 Mars 2013
http://www.ronenbekerman.com/photographic-approach-in-architectural-visualisation/

Article : Poétique, intelligent, engagé… Dix expériences qui montrent le jeu vidéo autrement
Le Monde, 30 septembre 2016
http://www.lemonde.fr/pixels/article/2016/09/30/poetique-intelligent-engage-dix-experiences-qui-montrent-le-jeu-video-autrement_5006009_4408996.html

Article : The Boomins World of Architecture That Only Exists in Pixels
Kelsey Campbell-Dollaghan, 7 Septembre 2015
http://gizmodo.com/the-booming-world-of-architecture-that-only-exists-in-p-1716656858

Article : The importance of visualizations in architecture
Andrew Brett, 7 Juillet 2013
http://www.renderthat.com/indicator/en/posts/the-importance-of-visualizations-in-architecture/

Article : Why architectural education in Britain is in need of repair
Olivier Wainwright, 30 Mai 2013
https://www.theguardian.com/artanddesign/architecture-design-blog/2013/may/30/architectural-education-professional-courses

Article : The Art of Rendering: How to Create "Arts & Crafts" Using 3ds Max, V-Ray and Photoshop
Ronen Bekerman, 2015
https://architizer.com/blog/the-art-of-rendering-how-to-create-arts-crafts/

Vidéo : Super mario - Low Poly
A+ Start, 8 Avril 2017
https://www.youtube.com/watch?v=A2gXyEyy_2U

Article : 3D Primer for Game Developers
David Silverman, 5 Mars 2013
https://gamedevelopment.tutsplus.com/articles/3d-primer-for-game-developers-an-overview-of-3d-modeling-in-games--gamedev-5704

Vidéo : PAUSE PROCESS, La lumière
PAUSE PROCESS, 23 Juillet 2015
https://www.youtube.com/watch?v=zyangpo2tN8

Vidéo : PAUSE PROCESS, La modélisation
PAUSE PROCESS, 12 Septembre 2015
https://www.youtube.com/watch?v=9Gh4YLh8xsU

Vidéo : PAUSE PROCESS, Le voxel
PAUSE PROCESS, 31 Mai 2015
https://www.youtube.com/watch?v=U-0n383c27E

Vidéo : PAUSE PROCESS, La texture
PAUSE PROCESS, 24 Juin 2015
https://www.youtube.com/watch?v=cslLlkybDDw

Article : PBR,La révolution silencieuse mais bien visible
Crabby, 21 Décembre 2014
https://www.gamekult.com/actualite/le-pbr-la-revolution-silencieuse-mais-bien-visible-142125.html

Article : PBR Texture Conversion
Joe "EarthQuake" Wilson, 1 Septembre 2015
https://www.marmoset.co/posts/pbr-texture-conversion/

Article : PBR Theory
OpenGL, Novembre 2016
https://learnopengl.com/#!PBR/Theory

Article : Crasher
Michael Vicente, 6 Janvier 2016
http://orbfolio.blogspot.fr/p/crasher-page.html

Article : Texture Types, Polycount
24 Mai 2015
http://wiki.polycount.com/wiki/Texture_types

Article : The Evolution of Architectural Visualization
6 Février 2015
https://www.vanguardstudio.com/blog/2015/2/5/the-evolution-of-architectural-visualization

Article : UV Mapping, Polycount
2 Novembre 2016
http://wiki.polycount.com/wiki/Uv

Document : Physically-Based Shading at Disney
Brent Burley, Walt Disney Animation Studios, 31 Août 2012

Présentation : Physically Based Lighting in Call of Duty: Black Ops
Dimitar Lazarov, Lead Graphics Engineer, Treyarch, SIGGRAPH 2011

Article : First Unity-built cartoon proves the engine isn't just for games
Derrick Rossignol, 24 Février 2017
https://www.engadget.com/2017/02/24/first-unity-built-cartoon-proves-the-engine-isnt-just-for-games/

Article : How to texture a videogame character
Nicolos Garilhe, 22 Juillet 2015
http://www.creativebloq.com/3d/how-texture-videogame-character-71515244

Article : Allegorithmic, la pépite clermontoise qui équipe les meilleurs jeux vidéo du monde
Le Monde, 29 Novembre 2016
http://www.lemonde.fr/pixels/article/2016/11/29/allegorithmic-la-pepite-clermontoise-qui-equipe-les-meilleurs-jeux-video-du-monde_5040141_4408996.html

Article :
Blenders New Principled Shader | Quick and Easy PBR in Cycles
Aidy Burrows, 13 Juin 2017
https://cgmasters.net/free-tutorials/blenders-new-principled-shader-quick-and-easy-pbr-in-cycles/

Article : Brief Considerations About Materials
21 Octobre 2010
http://www.manufato.com/?p=902

Article : Physically Based Texturing for Artists
Andrew Maximov, 2014
http://www.artisaverb.info/PBT.html

Article : What To Know When Creating Next Gen Assets
Aidy Burrows, 23 Avril 2015
https://cgmasters.net/free-tutorials/what-to-know-when-creating-next-gen-assets/

Article : Eevee Roadmap
Dalai Felinto, 22 Mars 2017
https://code.blender.org/2017/03/eevee-roadmap/

Documents : History and Comparative study of modern game engines
Partha Sarathi Paul, Surajit Goon, Abhishek Bhattacharya, 21 Mai 2012
https://www.researchgate.net/publication/236590476_HISTORY_AND_COMPARATIVE_STUDY_OF_MODERN_GAME_ENGINES

Article : How New Video-Game-Inspired Tools Are Redefining Post Occupancy Evaluation
Angus W. Stocking, 1 Novembre 2016
http://www.archdaily.com/798512/how-new-video-game-inspired-tools-are-redefining-post-occupancy-evaluation

Vidéo : PAUSE PROCESS, La caméra
PAUSE PROCESS, 4 Mars 2017
https://www.youtube.com/watch?v=3ymEX0PkZYk

Vidéo : PAUSE PROCESS, La lumière 2
PAUSE PROCESS, 23 Juillet 2015
https://www.youtube.com/watch?v=omvluLk25K8

Vidéo : PAUSE PROCESS, Les moteurs
PAUSE PROCESS, 5 Avril 2017
https://www.youtube.com/watch?v=D_pw841ZR1E

Document : A History of the Unity Game Engine
John Haas

Article : Virtual Reality for architecture, a beginner's guide
AECMagazine, 16 Février 2017
http://www.aecmag.com/59-features/1166-virtual-reality-for-architecture-a-beginner-s-guide

Article : Titre
Auteur + date
https://www.digitaltrends.com/computing/software-engineer-deus-ex-rendering/

Article : Ever wondered how a 3D game is rendered? Here's the scoop
Brad Bourque, 19 Mai 2015
Adresse Web

Article : Young Architect Guide: 9 Top Software Packages for VR Architectural Visualization
10 Juillet 2017
https://architizer.com/blog/young-architect-guide-vr-software/

Article : Making of: Project Soane - VR Lighting Experience
Jeff Mottle, 30 Mai 2017
http://www.cgarchitect.com/2017/05/making-of-project-soane---vr-lighting-experience

Article : How Physically Based Materials are Transforming Arch Viz
Jeff Mottle, 15 Août 2017
http://www.cgarchitect.com/2017/08/physically-based-materials-are-transforming-arch-viz

Article : Virtual Reality Uses in Architecture and Design
21 Janvier 2017
https://medium.com/studiotmd/virtual-reality-uses-in-architecture-and-design-c5d54b7c1e89

TABLE DES ILLUSTRATIONS

Figures 40 & 41
L'art de reprendre la main, conférence ENSANancy / François Robin / Mai 2016

Figures 42 & 43
Cutting Room: Joseph Kosinski talks to Archinect about his transition from architecture to Hollywood / Paul Petrunia / Septembre 2014
http://archinect.com/features/article/108112212/cutting-room-joseph-kosinski-talks-to-archinect-about-his-transition-from-architecture-to-hollywood

Figure 44
Image tirée du film «Mommy» / Xavier Dolan / 2014
http://www.semiozine.com/racines-carrees-mommy-peugeot

Figure 45
Vignettes d'extraits du film «The Grand Budapest Hotel» / Wes Anderson / 2015
http://www.delacritiquehysterique.com/the-grand-budapest-hotel-de-wes-anderson

Figure 46
Capture d'écran du jeu Half-Life 2, porté sur Unreal Engine 4
http://www.gamekyo.com/blog_article330483.html

Figure 47
Image tirée du film «Blade Runner» / Ridley Scott / 1982
http://1.bp.blogspot.com/-UysRSV5b-0g/U7kt7WGLTkI/AAAAAAAAMIA/V5M8PILdlsU/s1600/Blade_Runner_6.jpg

Figures 48 & 49
Photo personnelle / Exposition au musée des Arts Ludiques / 2016
Capture d'écran du jeu «Dishonored 2» / Arkane Studios / 2016
http://www.allocine.fr/film/court-metrage/news-18657240/

Figures 50 à 53
Team Fortress 2 / Valve / 2006
Journey / ThatGameCompany / 2012
Overcooked / Ghost Town Games / 2016
Assassin's Creed : Unity / Ubisoft / 2014

Figure 54
Capture d'écran du jeu «Super Mario Bros.», développé par Nintendo et sorti en septembre 1985 sur NES

Figure 55
Capture d'écran du jeu «Battlezone», développé par Atari et sorti en février 1998 sur Atari 2600

Figure 56
Capture d'écran du jeu «Starfox», développé par Nintendo et sorti en février 1993 sur SNES

Figure 57
Capture d'écran du jeu «Ultima Underworld», développé par Looking Glass Studios et sorti en 1992 sur DOS

Figure 58
Capture d'écran du jeu «Wolfenstein 3D», développé par iD Software et sorti en mai 1992 sur DOS

Figure 59
Extrait des sprites du jeu «Wolfenstein 3D»
https://www.spriters-resource.com/pc_computer/wolfenstein3d/sheet/65590/

Figure 60
Illustration des différentes consoles dites de «5eme génération» dans le courant des années 2000
https://www.timetoast.com/timelines/console-gaming-0da4de0f-8dc9-4dd5-baf1-8d4680fe9e0c

Figure 61
The Cararra Studio. Chapter 1: 3D Modeling Concepts and Techniques
http://www.webreference.com/3d/cararra/3.html

Figure 62
Wikipedia / Polygon Mesh
https://en.wikipedia.org/wiki/Polygon_mesh

Figures 63 & 64
Schémas personnels, inspirés de «3D Primer for Game Developers: An Overview of 3D Modeling in Games»

Figures 65 & 66
Captures d'écrans de l'émission «Super Mario - Low Poly (Evolution of Characters in Games) - Episode 1»
A+ Start / 8 Avril 2017
https://www.youtube.com/watch?v=A2gXyEyy_2U

Figure 67
Image .gif du forum reddit /r/horizon
https://www.reddit.com/r/horizon/comments/660jx6/heres_whats_happening_in_horizon_zero_dawn_every/

Figure 68
Wikipédia / UV Mapping
https://en.wikipedia.org/wiki/UV_mapping

Figure 69
Schémas personnels

Figure 70
Illustration d'un dépliage appliquée à un visage

Figure 71
Cylindrical UV Mapping (or Unwrap) with Relax solution and Blurry Texture
http://www.agisoft.com/forum/index.php?topic=705.0

Figure 72
Extrait du jeu «Crasher» / Michael Vicente
http://orbfolio.blogspot.fr/p/crasher-page.html

Figures 73 à 82
Textures de https://www.poliigon.com/
Rendu personnels

Figure 83
Texture du jeu Shuriken Game 2
https://samavan.deviantart.com/art/iPhone-Shuriken-Game-2-145376053

Figure 84
Texture personnelle

Figure 85
https://kyongame.com/blog/2017/1/19/subsurface-scattering-material-for-characters
https://static1.squarespace.com/c/5875793729687f9d49853844/58816981bebafbcb33797133/588169912994ca06fb122437/1484876177951/1698744cef46f09e0489e1faf06aef0a.jpg

Figure 86
Image extraite de l'article PBR Theory
https://learnopengl.com/img/pbr/microfacets_light_rays.png

Figure 87
Image extraite du site Allegorithmic
https://source.allegorithmic.com/img/logo-substance.png

Figure 88
Zoom sur Substance Painter, l'outil de painting / texturing
https://www.e-tribart.fr/blog/wp-content/uploads/2015/06/substancepainter1.jpg

Figure 89
Bibliothèque de Substance Painter, screenshot personnel

Figure 90
Adding Detail Textures, Unreal Engine documentation
https://docs.unrealengine.com/latest/images/Engine/Rendering/Materials/HowTo/DetailTexturing/DT_Hooked_Up_Textures.jpg

Figures 91 & 92
Images tirées de Blender, captures d'écrans personnelles

Figure 93
Schéma personnel

Figures 94 & 95
Images tirées de Blender, captures d'écrans personnelles

Figure 96
Schéma personnel

Figure 97
CryEngine Features
https://www.cryengine.com/files/features/categories/New_UI_4.jpg

Figure 98
https://4create.ru/games/47-unreal-engine-4.html
https://4create.ru/uploads/posts/2015-08/1438514111_scr1.jpg

Figure 99
Captures d'écran du jeu Pillow Castle
https://www.youtube.com/watch?v=HEBEQhwG-rU

Figure 100
Reddit, Capture d'écran du jeu Manifold Garden
http://i.imgur.com/4ILDVjg.png

Figure 101
Capture d'écran du jeu Portal
https://www.youtube.com/watch?v=AKtmAujwXRU

Figure 102
Capture d'écran du jeu Counter Strike Go
https://www.config-gamer.fr/media/k2/galleries/3704/p1_50403153411757.JPG

Figure 103
Image humoristique
http://www.ragecomic.fr/files/2012/09/fps-genius.jpg

Figure 104
Image tirée du jeu Splinter Cell Blacklist
http://www.dsogaming.com/wp-content/uploads/2013/08/modded-4.jpg

Figure 105
Image tirée du jeu Resident Evil 1
https://vignette1.wikia.nocookie.net/residentevil/images/5/59/Big_gallery.jpg/revision/latest?cb=20140827033522

Figure 106
Vidéo d'introduction de Metal Gear Solid Ground Zero
http://www.gamerheadlines.com/wp-content/uploads/2014/11/Screen-Shot-2014-11-05-at-09.53.45-1.jpg

Figure 107
Comparaison entre Mad Max et Metal Gear Solid V
https://pbs.twimg.com/media/C9oguH9XoAUDJiC.jpg

Figure 108
Aliasing / anti-aliasing
https://www.iguides.ru/upload/medialibrary/1ab/1ab71e7d175a33100f305ddd55e4ac60.jpg

Figure 109
Filtrage bilinéaire / trilinéaire
http://img.techpowerup.org/101222/texture%20filtering.jpg

Figure 110
Filtrage anisotropique
http://image.jeuxvideo.com/medias-md/146151/1461508992-3833-card.png

Figure 111
Bloom
http://www.mobygames.com/images/shots/l/342002-the-chronicles-of-spellborn-windows-screenshot-heavy-bloom.jpg

Figure 112
Ambient occlusion
http://i.imgur.com/BGAKd.jpg

Figure 113
Profondeur de champ
https://steamuserimages-a.akamaihd.net/ugc/542956332897548679/A07FDD4501D4631E441B14993DEE4EDCBB43AB5E/

Figure 114
Unreal Engine 4
https://4create.ru/uploads/posts/2015-08/1438514111_scr1.jpg

Figure 115
The Future : Blender 2.8
https://www.blender.org/wp-content/uploads/2017/08/thumbnail-2.png?x81735

Figure 116
Image tirée du jeu Firewatch
http://www.firewatchgame.com/screenshots/firewatch-e3-5.jpg

Figure 117
Image tirée du jeu Kerbal Space Program
https://pixelbandits.files.wordpress.com/2016/07/2862020-kerbalspaceprogram-3.jpg

Figure 118
Image tirée du jeu Cuphead
https://i.pinimg.com/originals/e5/90/be/e590beeef7e41c4deac07a1b53ce827c.jpg

Figure 119
Image tirée du jeu Monument Valley
http://static.mnium.org/images/contenu/videos/stars/Vignette_Pad_Emporter_-_Monument_Valley.jpg

Figure 120
Hirsch Log Homes, AEC
http://nvyve.com/portfolio-items/hirsh-log-homes/

Figure 121
Image tirée du jeu Age of Empire
https://i.ytimg.com/vi/eQKyFVP1f7k/maxresdefault.jpg

Figure 122
Image tirée du jeu Age of Empire Online
https://i.ytimg.com/vi/a_U-tA77f1Q/maxresdefault.jpg

Figures 122
Image tirée des jeux Battlefield Heroes et Battlefield 1
http://images.bit-tech.net/content_images/2009/07/battlefield-heroes-review/battlefield_heroes_11.jpg
http://sm.ign.com/ign_fr/screenshot/default/battlefield-1-screen-1024x570_cabh.jpg

Figure 124
Capture d'écran d'une première version de Unity
https://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf

Figure 125
Capture d'écran personelle

Figure 126
Image tirée du dessin animé Mr. Carton, France 4
http://studio-4.nouvelles-ecritures.francetv.fr/mr-carton-1451.html

Figures 127 à 130
Unity 5, captures d'écran personnelles

Figure 131
HDRi du Louvre, fichier personnel

Figures 132 à 144
Unity 5, captures d'écran personnelles

Figure 145
Illustration du HTC Vive et de l'Oculus Rift
https://images-na.ssl-images-amazon.com/images/I/81gCIwt39%2BL._AC_SL1500_.jpg
http://multimedia.bbycastatic.ca/multimedia/products/500x500/b00/b0008/b0008225.jpg

Figure 146
Capture d'écran d'Autodesk Live
https://damassets.autodesk.net/content/dam/autodesk/www/subscription/revit-live/fy18/features/images/bim-smart-large-1920x1073.jpg

Figure 147
Capture d'écran d'Enscape
https://apps.autodesk.com/RVT/fr/Detail/Index?id=2629595860167800202&appLang=en&os=Win64

Figure 148
Capture d'écran d'Autodesk Stingray
http://www.idcm.info/sites/default/files/images/Laatste%20nieuws/Stingray/stingray-stunning-rendering-and-visuals-large-1152x648.jpg

Figures 149 & 150
Capture d'écran du manuel du HTC Vive
https://loh9ewio6w1l7rxn2tnvzsoa-wpengine.netdna-ssl.com/wp-content/uploads/2016/05/htc-vive-chaperone-mode.jpg
https://www.vrnerds.de/wp-content/uploads/2016/01/htcvive-devkit-setupguide-02.jpg

Figure 151
Image tirée du jeu SuperHOT VR
https://s2-ssl.dmcdn.net/YdvrL.jpg

Figure 152
Image tirée du jeu Resident Evil VII
http://www.etr.fr/articles_images/356812-48.jpg

Figure 153
Image tirée du jeu Project Cars
https://www.macitynet.it/wp-content/uploads/2016/03/Project-Cars-VR-icon-1200-2-1024x523.jpg

Figure 154
Image tirée du jeu Project Cars
https://s.aolcdn.com/hss/storage/midas/4bd51ba05150f09e2ca67b994d26315a/203194790/Lucky%27s+Tale.png

Figures 155 à 160
SteamVR Locomotion and Teleportation Movement
https://unity3d.college/2017/05/16/steamvr-locomotion-teleportation-movement/

Figures 161 à 167
VR Interaction – SteamVR Lab Interaction System – Throwables
https://unity3d.college/2017/05/19/steamvr-lab-interaction-system-throwables/

LEXIQUE

DAO
Le dessin assisté par ordinateur (DAO) est une discipline permettant de produire des dessins techniques avec un logiciel informatique.

CAO
La conception assistée par ordinateur (CAO) comprend l'ensemble des logiciels et des techniques de modélisation géométrique permettant de concevoir, de tester virtuellement – à l'aide d'un ordinateur et des techniques de simulation numérique – et de réaliser des produits manufacturés et les outils pour les fabriquer.

BIM
BIM vient de l'anglais Building Information Modeling qui se traduit par Modélisation des Informations (ou données) du Bâtiment. Le BIM regroupe des méthodes de travail et une maquette numérique paramétrique 3D qui contient des données intelligentes et structurées. Le BIM est le partage d'informations fiables tout au long de la durée de vie d'un bâtiment ou d'infrastructures, de leur conception jusqu'à leur démolition.

Gameplay
Le gameplay, ou expérience de jeu, est un terme issu des jeux vidéo qui désigne l'ensemble des mécanismes utilisés pour augmenter le plaisir et la satisfaction de l'audience.

Octets
Groupe ou multiplet comprenant huit éléments binaires, souvent utilisé comme unité de base en informatique pour la longueur des mots, la taille des mémoires, etc.

Sprites
Un sprite, ou lutin, est dans le jeu vidéo un élément graphique qui peut se déplacer sur l'écran. En principe, un sprite est en partie transparent, et il peut être animé (en étant formé de plusieurs images matricielles qui s'affichent les uns après les autres)

5eme génération de consoles
L'histoire des consoles de jeux vidéo de cinquième génération représente l'ère des consoles 32 et 64 bits. Le marché est dominé par trois consoles, la PlayStation, la Saturn et la Nintendo 64.

Shader
Un shader ou nuanceur (le mot est issu du verbe anglais to shade pris dans le sens de « nuancer ») est un programme informatique, utilisé en image de synthèse, pour paramétrer une partie du processus de rendu réalisé par une carte graphique ou un moteur de rendu logiciel.

Addons
En informatique, un plugin ou plug-in, aussi nommé module d'extension, module externe, greffon, plugiciel, ainsi que add-in ou add-on en France, est un paquet qui complète un logiciel hôte pour lui apporter de nouvelles fonctionnalités.

Nodes
Terme traduisible par noeud, ou site.

TABLE DES MATIERES