CASQUE DE REALITE VIRTUELLE

Réalisation du prototype

Théau NICOLAS – Steven LAFOLIE

12/01/2017

IUT D'ALLIER
UNIVERSITÉ
Clermont Auvergne

UNIVERSITÉ
Clermont
Auvergne

SOMMAIRE

INTRODUCTION 3

PRESENTATION DU PROJET ET DU CAHIER DES CHARGES 3
SCHEMA FONCTIONNEL DU CASQUE 4
LES PROBLEMES VISUELS 4

PARTIE 1 – ETUDE PREALABLE 5

CHOIX DES COMPOSANTS 5
TEST DES COMPOSANTS 5
NOUVEAU COMPOSANT 6
CHOIX DES LOGICIELS ET PRESENTATION 7
TRINUS 7
UNREAL ENGINE 4 8
ARDUINO 10

PARTIE 2 - PROGRAMMATION DE LA CARTE (HARDWARE) 11

PROGRAMME UNOJOY : L'EXEMPLE MEGAJOYARDUINOSAMPLE.INO 13
LES PREMIERS CAPTEURS UTILISES 14
PROBLEME RENCONTRE : 15
NOUVELLE SOLUTION 16
UTILISATION D'UN NOUVEAU CAPTEUR : IMU-10DOF 16

PARTIE 4 – PROGRAMMATION (SOFTWARE) 19

LA PHASE D'INITIALISATION 19
TRAITEMENT DES DONNEES 20
GYROSCOPE 21
GYRO DRIFT FILTER 23
LE PROBLEME DU GYROSCOPE 23
LE PROBLEME DE L'ACCELEROMETRE 23

PARTIE 5 - ASSEMBLAGE ET CONCLUSION 24

RESULTATS FINAUX 26
CONCLUSION 26

ANNEXES **27**

POSITION **27**
AJOUT D'UNE MANETTE VIRTUELLE **27**
REALISATION AVEC VISUAL BASIC ET COMPATIBILITE **27**
BLENDER **28**

REALISATION D'UN CASQUE DE REALITE VIRTUELLE

Ce projet consiste à réaliser le prototype d'un casque de réalité virtuelle. La réalité virtuelle simule la présence physique d'un utilisateur dans un environnement créé ou généré par des logiciels. Elle permet de reproduire artificiellement une expérience visuelle et/ou auditive dans le cas de notre casque. Celui-ci est le moyen de matérialiser ce phénomène.

INTRODUCTION

Présentation du projet et du cahier des charges

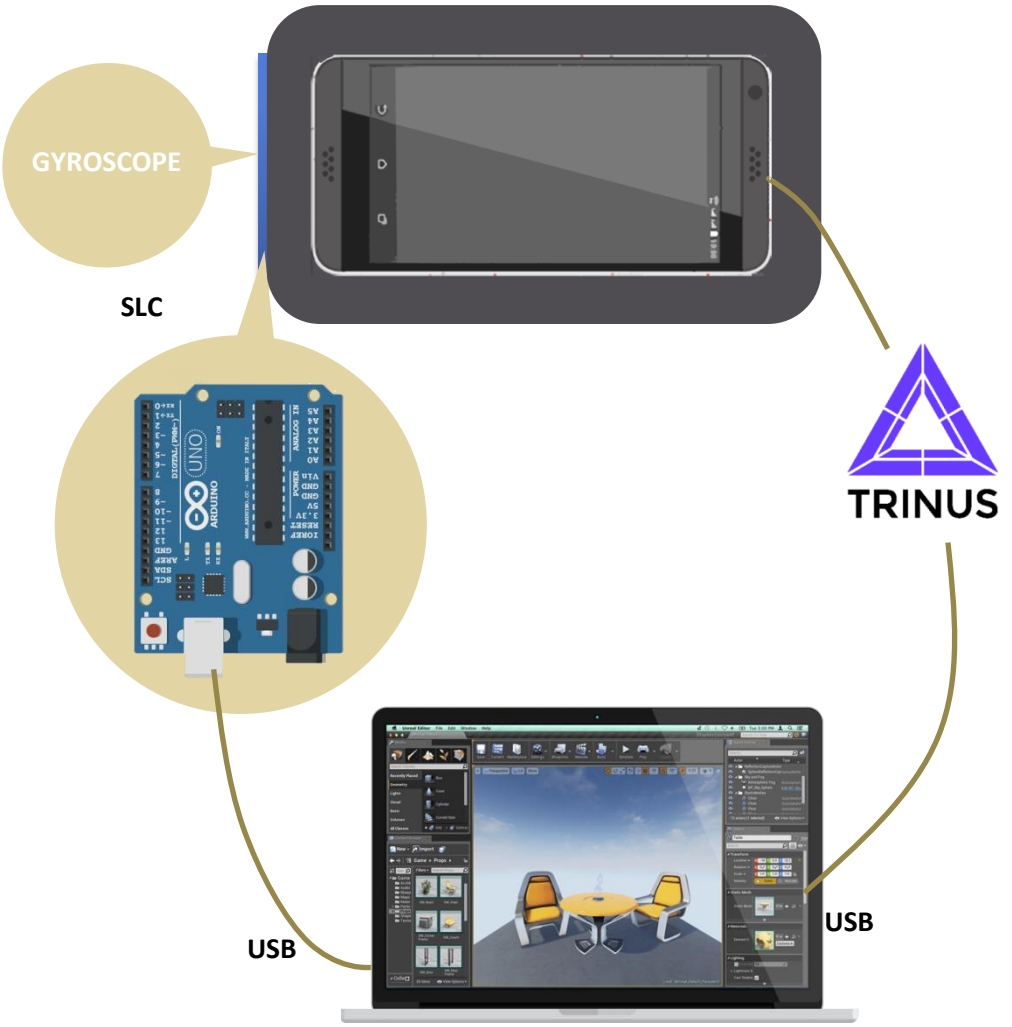
Un casque de réalité virtuelle classique est composé d'un écran, d'un système de positionnement et d'orientation, et d'un moyen de communication avec la source d'information. Actuellement, les casques présents sur le marché sont vendus à un prix compris entre 300€ et 1000 € environ. Pour notre part, nous souhaitons réaliser notre prototype avec un coût bien inférieur à celui des casques déjà existants dans le commerce. Nous avons donc opté pour la solution d'utiliser notre téléphone portable à la place d'écrans intégrés. Les smartphones déjà très performants, il existe une multitude d'applications qui permettent au téléphone de devenir un écran de réalité virtuelle (ex : Google Cardboard).

Le problème de ces applications, c'est qu'elles nécessitent une boussole et un accéléromètre intégrés au téléphone. Ces deux composants n'étant pas présents sur tous les modèles, même récents, nous avons décidé d'extérioriser la fonction de positionnement et de n'utiliser le téléphone qu'en tant qu'écran. Pour cela nous choisissons d'utiliser **ARDUINO** qui propose des composants de qualité nous permettant de réaliser simplement ces actions.

Le deuxième problème des applications dédiées sur smartphone, c'est que c'est le téléphone lui-même qui fait tourner ces programmes. Certains prennent beaucoup de place et de mémoire vive ce qui limite les performances de l'application. Nous souhaitons donc que ce soit notre ordinateur qui fasse tourner les applications. Nous décidons donc de projeter l'écran de l'ordinateur sur celui de notre smartphone.

Dans notre cas, la principale utilisation visée par notre projet est d'adapter notre casque pour les jeux vidéo. Nous utiliserons donc un moteur de jeux (**Unreal Engine 4**) qui est un environnement de programmation dédié aux jeux vidéo. Pour la communication entre l'ordinateur et le smartphone nous utiliserons l'application **Trinus** qui est gratuite et propose ce service.

Schéma fonctionnel du casque



Les problèmes visuels

Pour un effet d'immersion totale dans l'environnement virtuel, l'écran du téléphone portable doit être proche des yeux de l'utilisateur. L'humain n'ayant pas une vue adaptée pour voir d'aussi près, la réalisation du casque doit compenser ce problème. Pour cela on utilise des lentilles qui grossissent l'image. Mais là encore un problème persiste, c'est que, ayant deux yeux, la vue rapprochée nous parait doublée et il faudrait fortement « converger » pour parvenir à regarder l'écran. Donc nous devons aussi adapter le rendu visuel de l'écran afin que chaque œil regarde son propre écran :

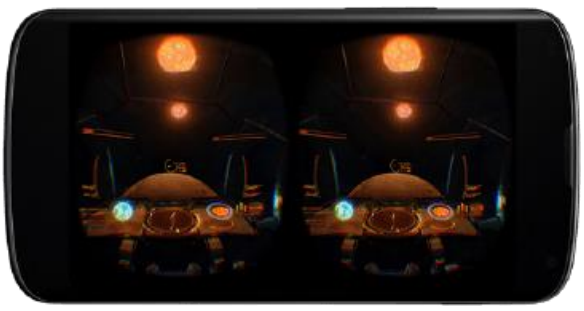


Figure 1 - Trinus VR

Steven LAFOLIE
Théau NICOLAS

PARTIE 1 – ETUDE PREALABLE

Choix des composants

Premiers composant et fonctionnement de base

Le choix des composants est porté sur des éléments compatibles avec ARDUINO, puisque les cartes de cette même marque proposent une multitude de fonctionnalités pour un prix très réduit. Notre premier choix a été de commander un module « Boussole » à 3 axes, et un module Accéléromètre. Le but de la boussole était de déterminer grâce à son fonctionnement l'angle de rotation du casque. Celui de l'accéléromètre était de déterminer la position du joueur dans l'espace afin de pouvoir se déplacer.

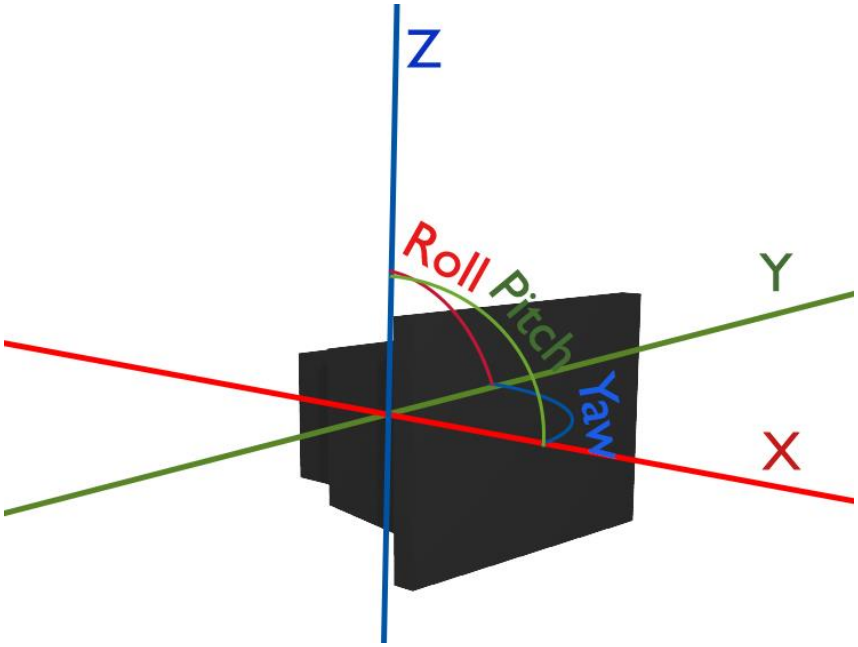


Figure 2 - Définition des axes de rotation

La boussole agit donc sur les éléments Pitch, Roll, et Yaw qui représentent la rotation dans l'espace de notre casque. Elle délivre une vitesse de rotation pour chaque angle. Ces valeurs une fois traitées sont assignées à la caméra virtuelle présente dans le jeu. On obtient donc normalement la même rotation dans le monde réel que dans le jeu ce qui donne l'illusion de s'y trouver.

L'accéléromètre, lui délivre une accélération. Une fois traitée on obtient un mouvement de la caméra virtuelle dans le jeu.

Pour définir correctement les valeurs nous définissons la rotation initiale a X=0, Y=0, et Z=0.

Test des composants

Résultats

Les valeurs sont cohérentes et la fluidité satisfaisante ainsi que le rendu sur écran qui est largement convenable pour un premier essai.

Problèmes

La boussole présente des perturbations au niveau magnétique, de plus, si elle gère très bien les rotations sur les axes X et Y, l'axe Z est influencé par ces deux derniers. Un offset aléatoire est également présent sur l'axe X.

L'accéléromètre est un composant très peu précis même dans produits les hauts de gamme. De plus, son fonctionnement ne permet finalement pas d'obtenir une position même en intégrant l'accélération car si l'objectif idéal est d'obtenir ces valeurs en déplaçant le capteur, la réalité est autre. En effet, il réagit à une rotation. Il est donc impossible d'en obtenir un mouvement linéaire.

Nouveau composant

La solution potentielle à ces problèmes est de les remplacer par un gyroscope qui intègre ces deux derniers éléments. La station permet de récupérer 10 types de données différentes :

- 3 données Accéléromètre
- 3 données Boussole
- 3 données Gyroscope
- 1 donnée Baromètre

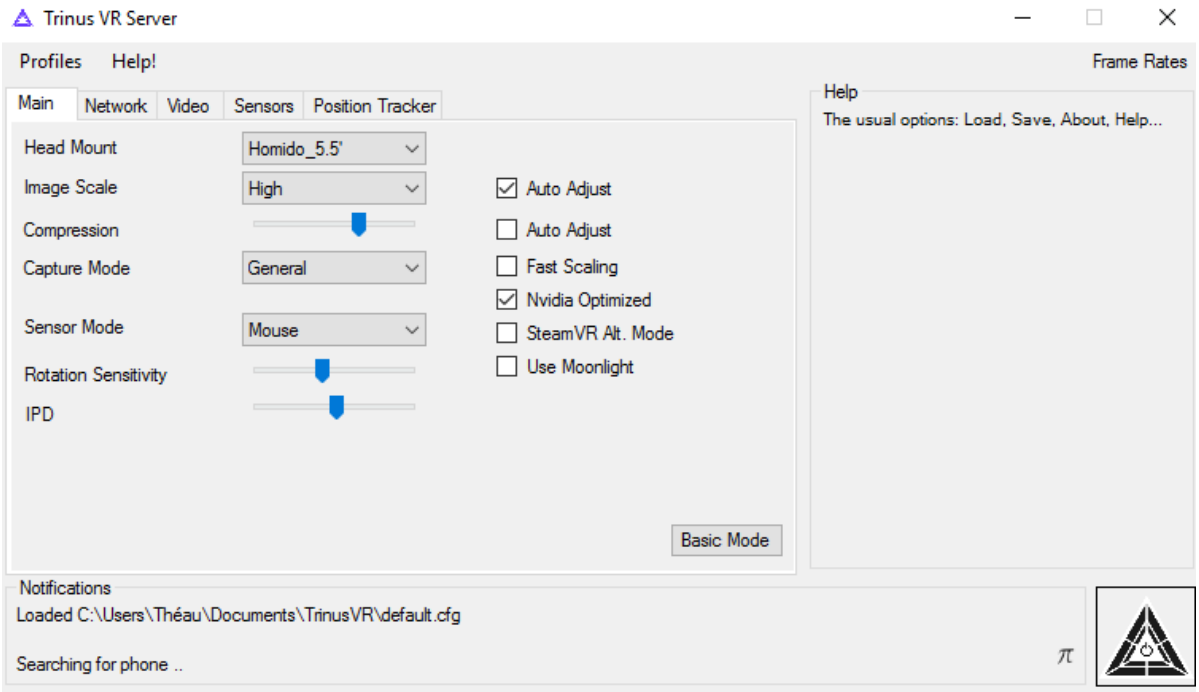
Nous récupérons donc les données du gyroscope et les transformons de façon à obtenir un angle afin d'obtenir un fonctionnement similaire à celui de la boussole.

Choix des logiciels et présentation

TRINUS

Trinus est une application smartphone qui propose aux utilisateurs de convertir leur téléphone en écran de réalité virtuelle depuis leur ordinateur, c'est-à-dire qu'elle adapte le rendu visuel de l'écran de PC à celui du smartphone tout en le séparant en deux pour que chaque œil puisse percevoir l'image correctement. Pour l'utiliser il faut télécharger l'application smartphone mais aussi le serveur sur PC.

Le logiciel se présente comme ceci :



On peut y régler tous les paramètres et options de réseaux de port USB et d'affichage.

Le logiciel propose plusieurs liens de communications comme le Wifi ou le Bluetooth, mais celui qui présente le moins de défauts reste l'USB.

UNREAL ENGINE 4

L'Unreal Engine est un moteur de jeux professionnel très puissant qui permet à son utilisateur de programmer, créer, assembler et animer des environnements virtuels aussi réalistes qu'imaginatifs. Il est entièrement gratuit à l'utilisation non commerciale ce qui nous permet de l'utiliser dans notre projet. Le domaine des jeux vidéo est en constante expansion et grâce aux outils développés par les entreprises et aux communautés amatrices et professionnelles qui entoure l'Unreal, il est actuellement possible de créer de petits jeux vidéo avec une qualité satisfaisante avec très peu de moyens. (Cf : Youtube - Substance designer UE4)

Figure 3 – Performances de l'UE4

Le logiciel nous propose deux langages de programmation :

- Le C++ langage de programmation orientée objet classique qui doit être développé et compilé dans un logiciel extérieur comme Visual Studio par exemple.
- Le Blueprint qui est un langage de programmation graphique intégré. Bien qu'un peu plus limité par rapport au C++, il assure une compatibilité totale avec le moteur et permet de réaliser rapidement et surement des actions simples telles que des animations, des évènements ou des déplacements dans l'espace.

Comme la programmation de notre casque est en partie intégrée dans le jeu, nous choisissons d'utiliser le langage Blueprint pour pouvoir réaliser des tests simples rapidement.

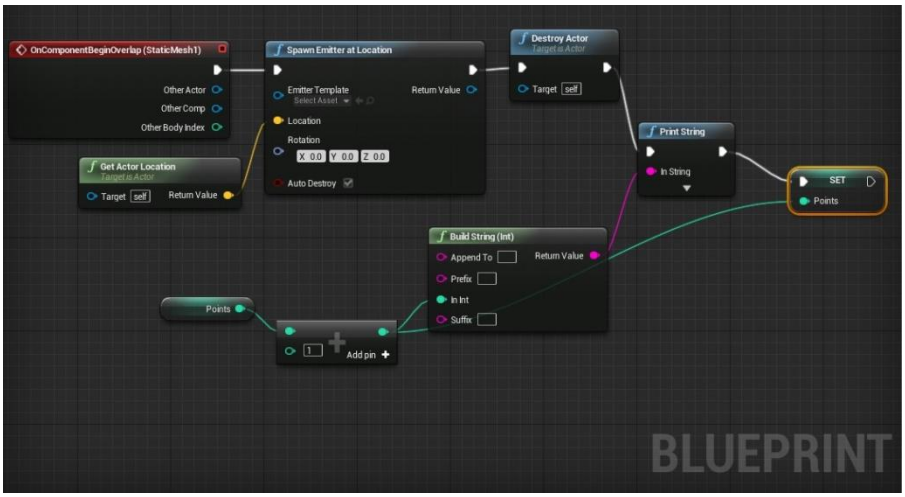


Figure 4 - Blueprint editor

Steven LAFOLIE
Théau NICOLAS

Le Langage graphique est géré par le logiciel sous forme de nodes. C'est un langage qui gère tous les effets visuels des jeux vidéo, il est très rapide parce qu'il est exécuté directement par la carte graphique. Les deux principaux langages existants sont l'OpenGL et DirectX (Développé par Microsoft). Ces langages permettent de définir ce qu'on appelle des matériaux et de leur appliquer des textures.

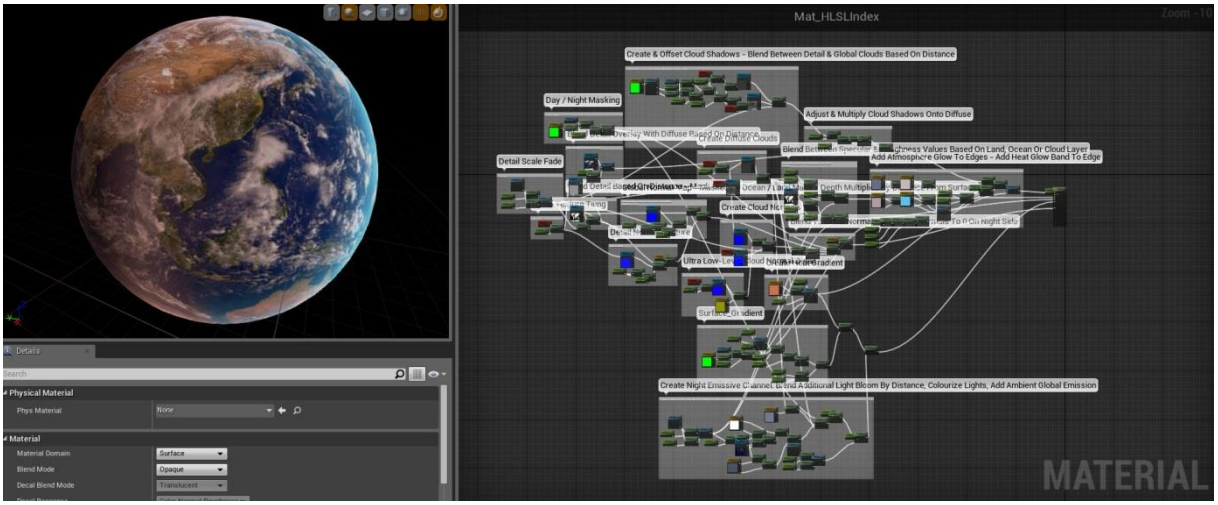


Figure 5 - Material editor

Dans les différents moteurs de jeux existants, on peut créer des objets graphiques en 2D ou en 3D. L'Unreal Engine offre les deux possibilités. Pour intégrer un objet graphique, il faut d'abord le créer, le modéliser, le texturer correctement afin de l'importer dans le moteur. Selon la nature de l'objet (2D ou 3D) on utilisera des logiciels différents (Blender pour la 3D, et Gimp pour la 2D ou les textures pour les logiciels OpenSource). Il suffit ensuite de les importer dans l'Unreal pour pouvoir les programmer et les intégrer au jeu.

Toutes ces étapes demandent un temps de fabrication considérable et des compétences très diverses. La création d'un objet immobile demande très peu de ressources alors qu'un élément mobile, comme un personnage, qui est déjà très difficile à modéliser, demande beaucoup plus de travail comme l'animation ou la programmation d'intelligence artificielle même simple.

ARDUINO

Arduino est le nom d'un « fabricant » de circuits imprimés sur lesquels il est possible de brancher toutes sortes de capteurs (luminosité, contact, pression, température etc...) et actionneur (LED, moteur, relais …). Cette carte se programme sur l'ordinateur via un câbles USB (ou autre selon le type de carte comme la Arduino Severino) et permet de commander n'importe quel appareil. Il suffit pour cela de modifier le code qu'exécute l'Arduino. La particularité de ce système est qu'il est libre, c'est-à-dire que les plans des cartes sont disponibles gratuitement. Il est possible de modifier et de réutiliser ces plans pour pouvoir créer sa propre carte. Seul le nom « Arduino » est déposé et n'est pas utilisable.

Figure 6 - Carte Arduino UNO qui est la plus repandue

Il existe divers types de cartes Arduino (Arduino Mega, Duemilanove, Nano...) qui sont toutes programmables avec le même IDE qui est propre à Arduino.

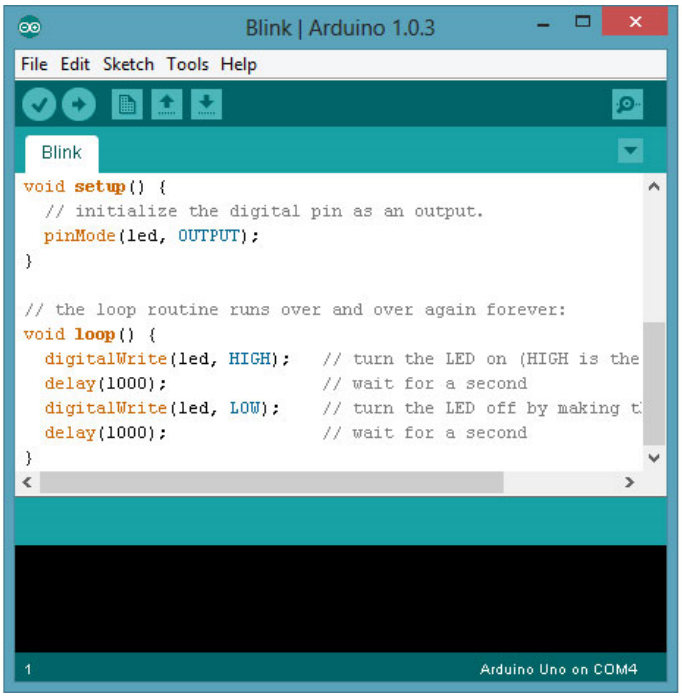


Figure 7 - Interface de programmation Arduino

Pour la réalisation de ce projet nous décidons donc d'utiliser une carte Arduino Uno.

PARTIE 2 - PROGRAMMATION DE LA CARTE (HardWare)

Nous partons dans un premier temps sur l'utilisation d'un programme OpenSource qui est UnoJoy. Ce programme est conçu pour la réalisation d'une manette pour console de jeu vidéo (en particulier playstation 3) ou pour PC. Il peut être par exemple utilisé pour réaliser un simulateur de vol par exemple. UnoJoy permet donc de transformer le petit module expérimental Arduino Uno en interface joystick HID (Human Interface Device) sous Windows et sur PS3.

On retiendra d'UnoJoy les dossiers et fichiers suivants :

- Des dossiers regroupant les programmes principaux pour différentes cartes Arduino avec les fichiers Headers (DoubleJoy, LeoJoy, MegaJoy, UnoJoy)
- Un dossier servant à l'installation des drivers pour pouvoir reconnaître les cartes Arduino converties en Joystick (Driver)

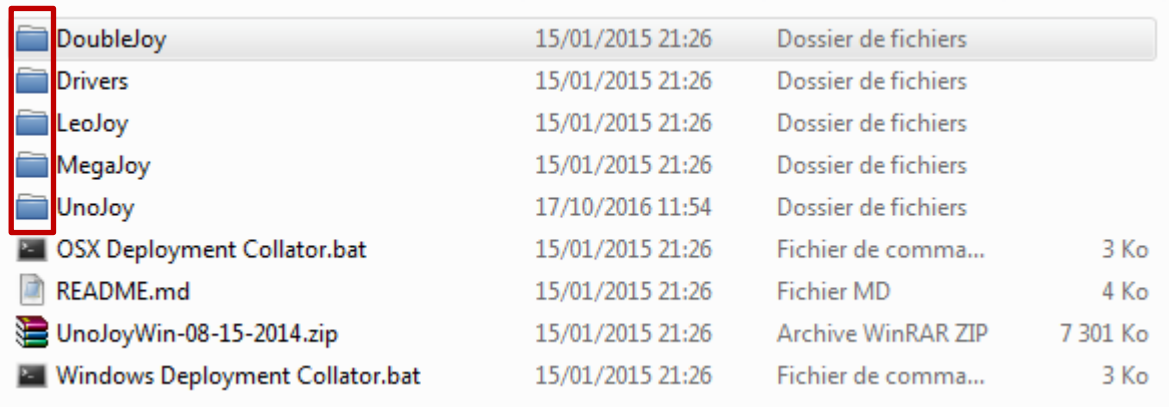
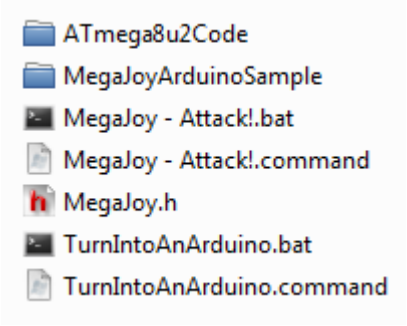


Figure 8 - Dossier d'installation des drivers

Prenons par exemple le dossier MegaJoy :



Il contient :

- MegaJoyArduinoSample : dossier contenant le programme .ino pour la carte Arduino
- MegaJoy - Attack!.bat : Une commande permettant de convertir la carte Arduino en carte de commande joystick (la carte est donc reconnue comme étant un joystick et non une carte Arduino).
- TurnIntoAnArduino.bat : Donne l'effet inverse de MegaJoy - Attack!.bat (la carte sera reconnue comme étant une carte Arduino).
- Atmega8u2Code : Sert à la reprogrammation du chip Série/USB de type ATmega8u2

Pour pouvoir transformer la carte Arduino il est nécessaire de la mettre en mode DFU (Device Firmware Update) qui va servir à reprogrammer le microC de communication Série/USB. Seules les cartes Arduino possédant un chip de communication de type ATmega8u2 ou ATmega16u2 peuvent être reprogrammées de cette façon (autrement dit les cartes Arduino officiel).

On envoie dans un premier temps le programme dans la carte Arduino grâce à l'IDE.
Pour convertir la carte Arduino il suffit juste de connecter la carte Arduino au PC, d'ouvrir l'un des 2 fichiers .bat (selon la conversion voulu) :

Une fois la conversion terminée, il faut déconnecter et reconnecter la carte Arduino.

Steven LAFOLIE
Théau NICOLAS

Programme UnoJoy : l'exemple MegaJoyArduinoSample.ino

Tous les programmes UnoJoy sont les mêmes peu importe la carte. Le programme MegaJoyArduinoSample.ino est donc un exemple de ce que l'on peut faire avec UnoJoy sur une carte Arduino Mega.

Dans le void setup() on initialise tous les ports numériques de l'Arduino comme étant des commandes TOR (grâce à la fonction setupPins() que l'on programme ensuite) et tous les ports analogiques comme étant des commandes analogiques donc des joystick (grâce à la fonction setupMegaJoy() qui est une fonction propre à UnoJoy et configurée dans le fichier hearder MegaJoy.h).

```
void setup()
  setupPins();
  setupMegaJoy();
}

void loop(){
  // Always be getting fresh data
  megaJoyControllerData_t controllerData = getControllerData();
  setControllerData(controllerData);
}

void setupPins(void){
  // Set all the digital pins as inputs
  // with the pull-up enabled, except for the
  // two serial line pins
  for (int i = 2; i <= 54; i++){
    pinMode(i, INPUT);
    digitalWrite(i, HIGH);
  }
}
```

Le programme principal en lui-même est très court peu importe la version de carte utilisé ou le nombre de port utilisée :

```
void loop(){
  // Always be getting fresh data
  megaJoyControllerData_t controllerData = getControllerData();
  setControllerData(controllerData);
}
```

megaJoyControllerData_t controllerData = getControllerData();

On récupère les données fournies par les joysticks (mouvement d'un joystick ou appui sur un bouton) et on va comparer toutes les données de façon à savoir quel module (bouton ou autre) est actionné pour être ainsi reconnu comme étant un « mouvement » spécifique de la « manette ».

La fonction *analogRead(pin)* renvoie une valeur codée sur 10 bits, or on veut obtenir des données su 8 bits, on utilise donc *controllerData.analogAxisArray[0]* qui est une fonction propre d'UnoJoy et qui réalisera cette conversion pour les pins analogiques.

```
controllerData.analogAxisArray[0] = analogRead(A0);
controllerData.analogAxisArray[1] = analogRead(A1);
controllerData.analogAxisArray[2] = analogRead(A2);
controllerData.analogAxisArray[3] = analogRead(A3);
controllerData.analogAxisArray[4] = analogRead(A4);
controllerData.analogAxisArray[5] = analogRead(A5);
controllerData.analogAxisArray[6] = analogRead(A6);
controllerData.analogAxisArray[7] = analogRead(A7);
controllerData.analogAxisArray[8] = analogRead(A8);
controllerData.analogAxisArray[9] = analogRead(A9);
controllerData.analogAxisArray[10] = analogRead(A10);
controllerData.analogAxisArray[11] = analogRead(A11);
```

On renvoie ensuite les données du contrôleur avec `return controllerData;`

Les premiers capteurs utilisés

Pour réaliser notre casque VR nous devons prendre en compte les différents axes de rotations de la tête sur X,Y et Z et de translation.
Pour pouvoir récupérer ces mouvements nous partons donc sur l'utilisation de 2 capteurs qui sont une boussole (ou compas aussi) HMC5883L et un accéléromètre MMA7361

Les capteurs et UnoJoy

L'accéléromètre nous renvoie une valeur analogique pour chaque axe. La programmation étant plus simple à réaliser, nous nous penchons donc dans en premier temps sur la programmation de la boussole et son adaptation au programme d'UnoJoy.
En effet ce module nous montre physiquement une pin SCL et une pin SDA. Les données sont donc transmises par communication de bus I²C.

Sur Arduino, une librairie existe et est fournie d'origine avec l'IDE lors de l'installation. Il s'agit de la librairie « Wire ». Pour utiliser les fonctions de cette librairie il est donc nécessaire de l'inclure dans le programme UnoJoy grâce au fichier headers *Wire.h*.
Il existe aussi une librairie OpenSource pour la boussole HMC5883L que nous avons téléchargée au préalable. Comme pour la librairie Wire, il nous faut aussi ajouter les fichiers headers de la librairie du HMC5883L pour pouvoir utiliser les fonctions correctement.

```
#include <Wire.h>   //Librairie Wire pour la communication I2C
#include <Adafruit_Sensor.h>
#include <Adafruit_HMC5883_U.h>

void setup(){
  Serial.begin(9600);
  Wire.begin();  //Initialisation de la librairie Wire
  //setupPins();
  setupUnoJoy();
  dataForController_t controllerData = getBlankDataForController();
}
```

D'après la librairie du HMC5883L, les valeurs envoyées par chaque axe varient entre - 800 et + 800. Le problème est que les données de contrôleur (controllerData) sont codées sur 8 bits donc de 0 à 255, nous devons donc procéder à un mappage des valeurs envoyées par le capteur.

```
int X = event.magnetic.x;
int Y = event.magnetic.y;
int Z = event.magnetic.z;

X = map(X,-800,800,0,255);
Y = map(Y,-800,800,0,255);
Z = map(Z,-800,800,0,255);

Serial.print("X: "); Serial.print(X); Serial.print("  ");
Serial.print("Y: "); Serial.print(Y); Serial.print("  ");
Serial.print("Z: "); Serial.print(Z); Serial.println("  ");

// Set the analog sticks
//  Since analogRead(pin) returns a 10 bit value,
//  we need to perform a bit shift operation to
//  lose the 2 least significant bits and get an
//  8 bit number that we can use
//controllerData.leftStickX = analogRead(A0) >> 2;
controllerData.leftStickY = Z;
controllerData.rightStickX = Y;
controllerData.rightStickY = X;
//controllerData.rightStickZ = Zmagnetic;
// And return the data!
return controllerData;
```

Problème rencontré :

Les valeurs sont envoyées correctement mais lors du passage de la carte Arduino Uno en mode Joystick, ses données sont erronées et nous ne savons pas exactement d'où cela peut provenir. Nous abandonnons donc l'idée d'utiliser UnoJoy.

Nouvelle solution

L'intégration d'une communication I²C n'ayant pas réussi avec UnoJoy, nous décidons de renvoyer nos données fournies pas le capteur directement sur le port série et de le récupérer par la suite grâce à un programme que nous créerons spécialement et qui servira d'intermédiaire aux jeux VR et à Arduino.

Nous lisons donc dans un premier temps les données en X, Y et Z renvoyées par le HMC5883L.

```
Xmagnetic = HMC5803L_Read(X);  //lecture sur 3 axes et sortie sur le port sériel
Ymagnetic = HMC5803L_Read(Y);
Zmagnetic = HMC5803L_Read(Z);
```

Nous réalisons ensuite un mappage sur 360 (représentant les 360° de chaque axe).
Un nouveau problème se présente : présence d'un offset. Nous essayons donc de corriger cette offset directement sur le programme qui lie les données envoyées par la carte.

Utilisation d'un nouveau capteur : IMU-10DOF

Figure 9 - Station gyroscopique IMU-10DOF

Précédemment nous avons pu constater que l'utilisation d'une boussole (un compas) ne donnait pas exactement ce que l'on voulait obtenir. En effet, le capteur le plus adapté pour les mouvements de rotation de la tête sur 3 axes est le Gyroscope. Nous partons donc sur l'utilisation du module IMU-10DOF qui contient divers capteurs (compas, température, baromètre...) dont un module faisant office d'accéléromètre et de Gyroscope en même temps : le MPU9250. Il est aussi plus précis que les capteurs précédemment utilisés.
Ce modules utilise aussi la communication I²C et possède sa librairie de fonction.

L'accéléromètre, par exemple, nous renvoie des valeurs variant de – 16383 à + 16384 d'après le fichier headers.

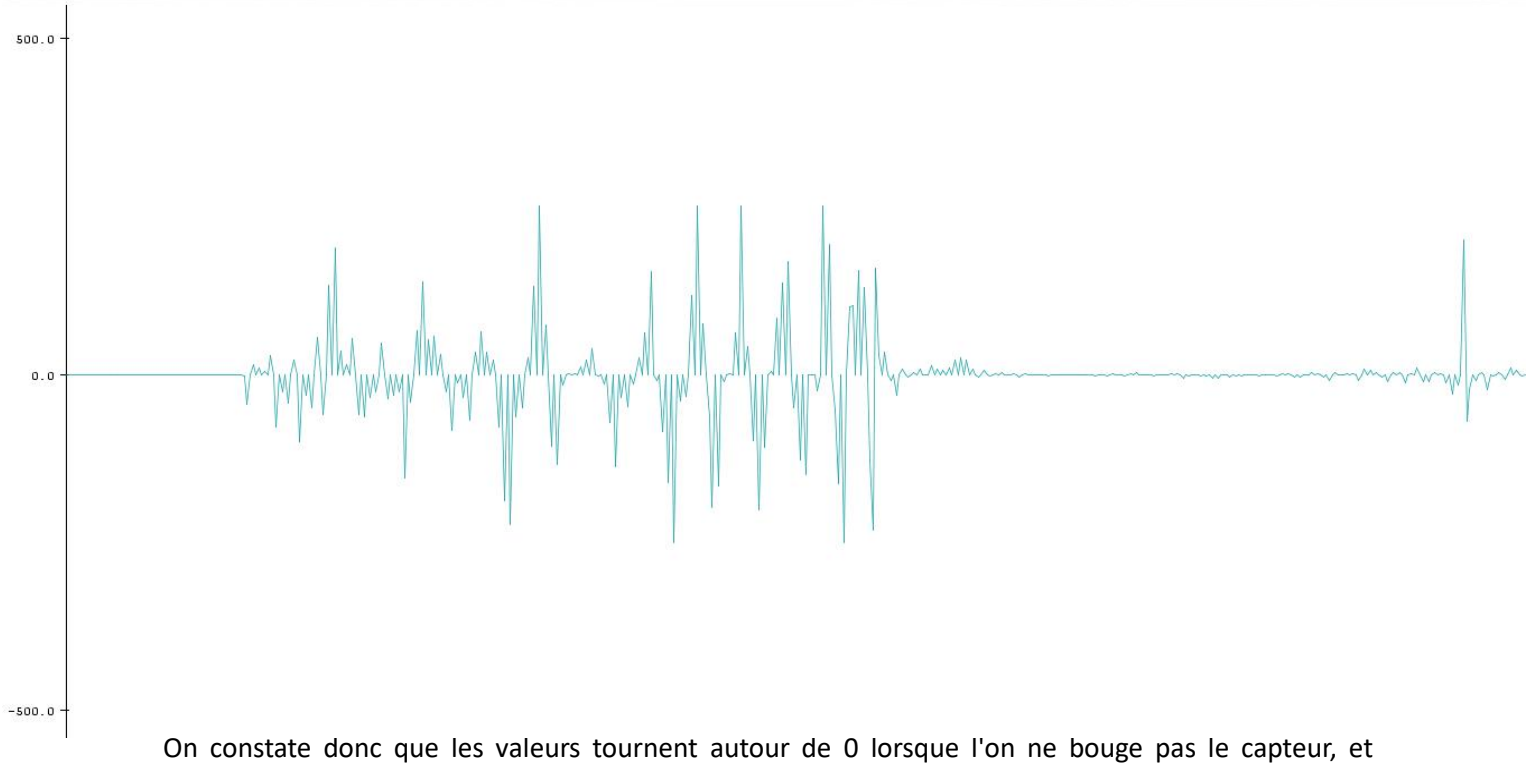
```
void getAccel_Data(void)
{
    accelgyro.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz);
    Axyz[0] = (double) ax / 16384;
    Axyz[1] = (double) ay / 16384;
    Axyz[2] = (double) az / 16384;
}

void getGyro_Data(void)
{
    accelgyro.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz);
    Gxyz[0] = (double) gx * 250 / 32768;
    Gxyz[1] = (double) gy * 250 / 32768;
    Gxyz[2] = (double) gz * 250 / 32768;
}
```

Grâce à la librairie du capteur CPU9250, on peut réaliser le transfert des données des axes du gyroscope et de l'accéléromètre sur le port série. La librairie répartie chaque axe dans un tableau de données par 3. Si l'on met donc 0 dans Gxyz[], on récupère alors les valeurs de l'axe X du gyroscope, 1 pour Y et enfin 2 pour Z . Il en va de même pour chaque axe de l'accéléromètre.

```
Serial.print("");
Serial.print(Gxyz[0]);
Serial.print(" ");
Serial.print(Gxyz[1]);
Serial.print(" ");
Serial.print(Gxyz[2]);
Serial.print(" ");
Serial.print(Axyz[0]);
Serial.print(" ");
Serial.print(Axyz[1]);
Serial.print(" ");
Serial.println(Axyz[2]);
```

Arduino IDE nous propose aussi de visualiser en temps réel, soit les valeurs envoyées sur le port série grâce au moniteur série, ou tout simplement de visualiser l'évolution des valeurs graphiquement grâce au traceur série. On peut donc relever par exemple le graphique suivant qui est la visualisation de l'axe X du gyroscope.

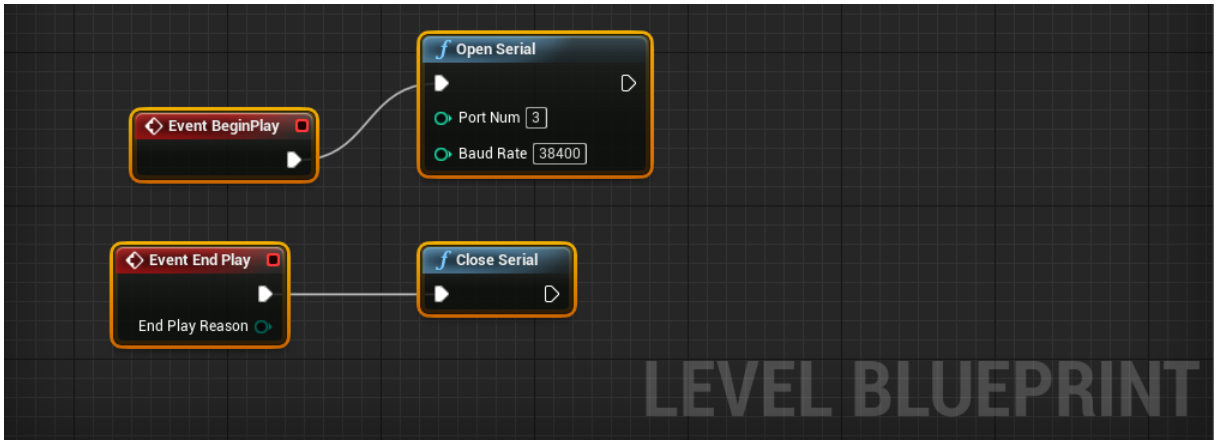


On constate donc que les valeurs tournent autour de 0 lorsque l'on ne bouge pas le capteur, et qu'elles sont positives lorsque l'on part dans un sens, puis négatives dans le sens opposé.

PARTIE 4 – PROGRAMMATION (SoftWare)

La programmation du casque sur ordinateur se base sur les données récupérées depuis le port COM. Pour cela on définit le numéro du port sur lequel aller chercher les informations. Une fois défini on ouvre le port. Une fois le jeu fini le port COM se ferme.

L'Unreal Engine propose un espace de programmation appelé « level blueprint » qui permet d'y coder tous les éléments qui agissent sur les données du jeu comme les paramètres généraux ou les données externes.



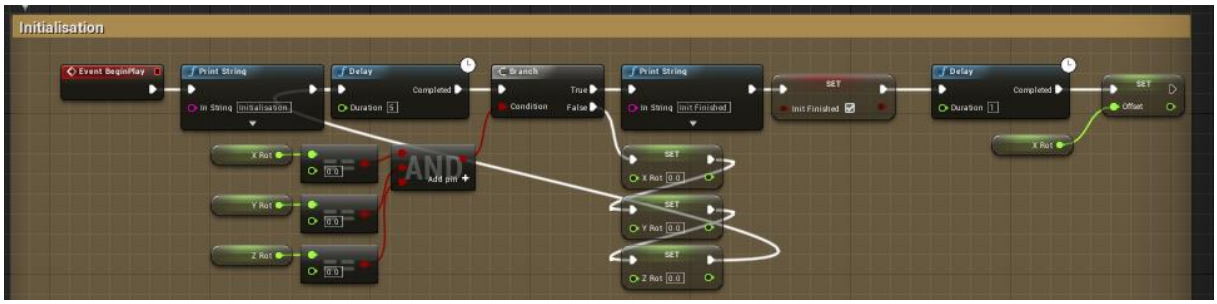
Une fois la gestion du port effectuée, il faut à présent traiter les données. Le programme se déroule en trois étapes :

- Phase d'initialisation
- Récupération et traitement des données
- Application aux éléments virtuels

La phase d'initialisation

En testant le programme, nous nous sommes rendu compte que la carte a besoin d'un temps d'initialisation avant de transférer des valeurs correctes. Le problème est que, du fait de notre technique de programmation et des informations récupérées pendant ce temps, le logiciel ne traite pas correctement les valeurs et dévie de sa rotation initiale. Lorsque la phase est terminée, on se retrouve avec un offset aberrant et le programme devient injouable.

Pour pallier à ce problème, nous avons défini un temps pendant le lequel on ne traite pas les valeurs et à la fin duquel la valeur de rotation est automatiquement redéfinie à 0. Ce qui assure d'obtenir la rotation initiale au début du jeu. Aucune rotation initiale n'as été définie en dehors du logiciel, la position du casque en début de jeu n'influe pas sur celle de la caméra virtuelle.



Steven LAFOLIE
Théau NICOLAS

Algorithme :

```
DEBUT
Fc.DébutdeJeu(Event Begin Play)
{
        AFFICHER(“Initialisation”) ;
        DELAI T=5s
        SI (X==0 & Y==0 & Z==0)
        ALORS
        {
                AFFICHER(« Fin Initialisation ») ;
                SET InitialisationFinie = True ;
        }
        SINON
        {
                SET X=0 ;
                SET Y=0 ;
                SET Z=0 ;
        }
}
FIN
```

Traitement des données

Pour commencer, les données délivrées par la carte ARDUINO sont de type String, il faut donc les convertir. On sépare les données de façon à récupérer le X, le Y, et le Z selon la forme qu’elles prennent :

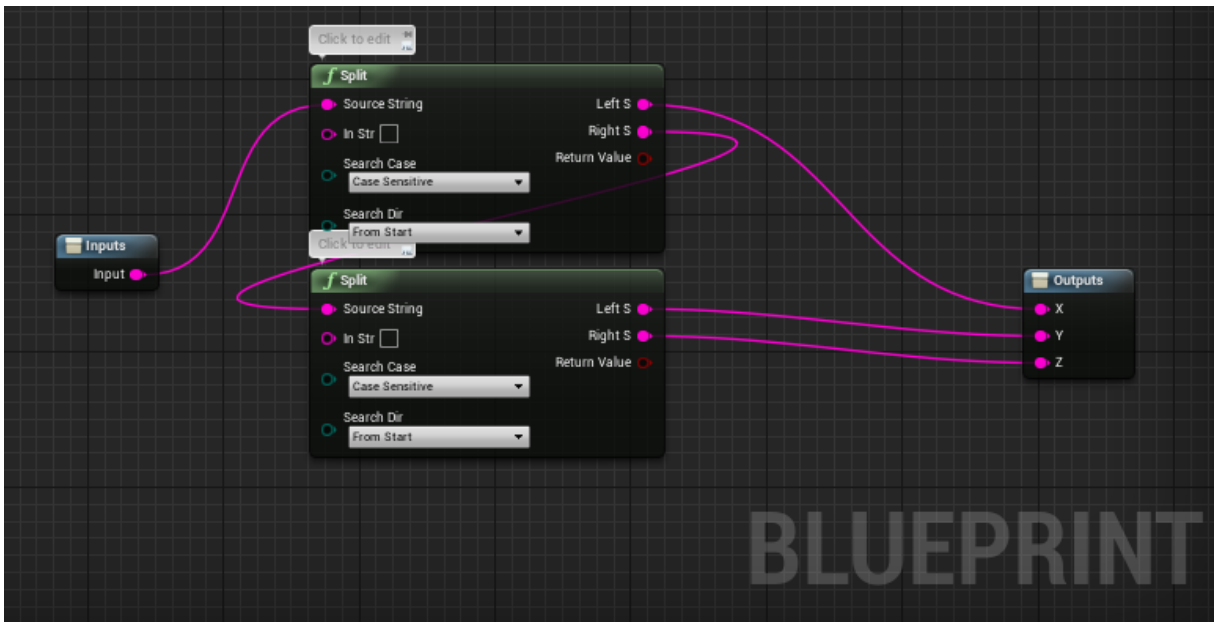
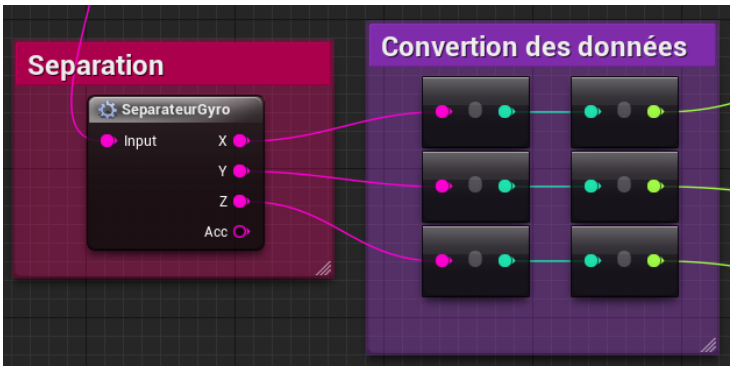


Figure 10 - Macro Fonction Séparation

Steven LAFOLIE
Théau NICOLAS

On peut ensuite convertir les données afin de les traiter. Une conversion en INT préalable permet de réduire les parasites dus aux variations magnétiques de la carte. Le type FLOAT étant plus adapté pour manipuler ces données, nous les convertissons une nouvelle fois :



COULEURS :

- String
- Int
- Float
- Booléen

Gyroscope

Pour l'instant, les informations récupérées sur le port correspondent à la vitesse de rotation du gyroscope. Le but de notre programmation est de définir une position en fonction de celle du gyroscope. On doit donc intégrer la vitesse afin d'obtenir une position.

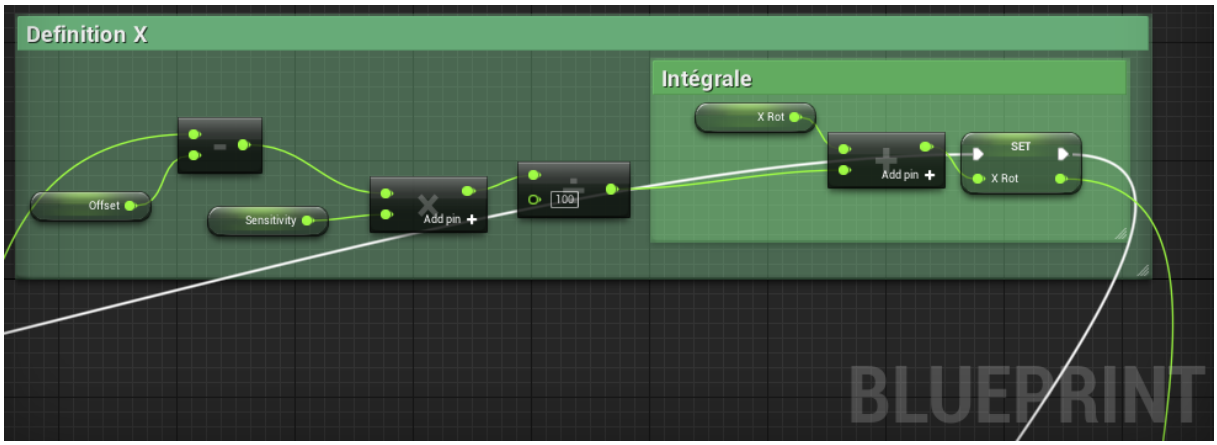
La fonction d'intégration n'est pas présente dans le langage de programmation et aucune fonction ne permet de la coder de façon exacte, mais il existe une méthode simple qui donne une approximation du comportement de l'intégrale avec une erreur minime :

$$P(t) = \int_0^t v(t)dt \approx \sum_0^t v(t)Ts$$

Avec Ts intervalle de temps entre chaque valeur du gyroscope (Période).

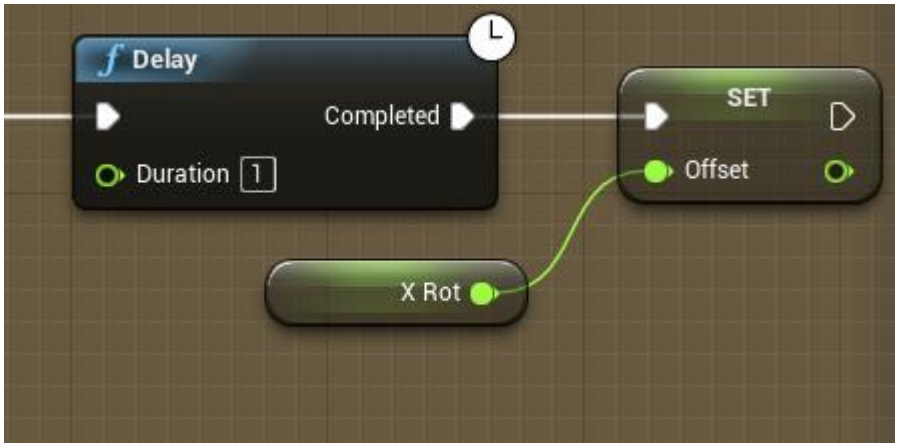
Et P = 0 si t = 0

Grâce à cette méthode, on obtient bien le fonctionnement primaire de notre casque c'est-à-dire que si on l'applique aux valeurs de X, Y, et Z la rotation de la caméra virtuelle correspond à celle du gyroscope :



La valeur de X est d'abord traitée de façon à éviter que l'addition successive des valeurs n'entraine une rotation involontaire de la caméra. La valeur du gyroscope bien que définie sur 360° présente un coefficient dérivatif lorsqu'on la reçoit dans le logiciel. On la multiplie donc par un coefficient de sensibilité qui calibre la caméra virtuelle sur 360°. Ce coefficient est défini par des tests successifs afin d'avoir un rendu optimal.

On observe par la suite qu'après la phase d'initialisation, les valeurs d'Y et de Z sont bien remises à zéro et de façon précise. La valeur de X présente un petit offset qui engendre cette rotation dont nous ne voulons pas. On récupère donc cette valeur juste après la phase d'initialisation et on la soustrait continuellement à la valeur de X.



Du fait de l'approximation, des erreurs de positions apparaissent. Lorsque les valeurs du gyroscope changent plus rapidement que la fréquence de récupération d'informations, on ne détectera pas cette variation et l'intégrale approximative délivrera une valeur faussée. Ce phénomène est appelé Drift. Il est responsable du fait que lorsque que l'on souhaite revenir à la position initiale (X=0, Y=0, Z=0), on observe une légère déviation qui vient perturber le bon fonctionnement du casque. C'est pour cela qu'il est important de choisir une bonne fréquence de récupération d'informations. La fréquence recommandée pour les éléments mécaniques varie entre 100Hz et 200Hz.

Même avec une fréquence correctement choisie, le Drift opère encore. Il faut donc lui appliquer un filtre pour le corriger.

Gyro drift filter

Le Gyro Drift Filter est le filtre qui va corriger notre valeur de gyroscope.

Le problème du gyroscope

Le gyroscope est un composant qui n'est pas parfait et qui réagit à beaucoup de phénomènes parasites. Il apparait très précis sur une courte durée mais très peu sur une longue, ce qui explique notre problème de dérivation. En exploitant le potentiel de notre carte on sait qu'elle possède délivre des données d'accéléromètre. Celles-ci vont nous être utiles dans la correction de nos valeurs.

Le problème de l'accéléromètre

L'accéléromètre, même très cher est un composant lui aussi très sensible aux phénomènes étrangers et de ce fait il est très peu précis sur une courte durée. Il est alors impossible de l'exploiter pour récupérer des données de rotation en temps réel. Mais contrairement au gyroscope, il est précis sur la longueur, c'est-à-dire qu'il ne dérive pas. Mais pour l'utiliser il faudra lui appliquer un filtre passe-bas. Lorsque le gyroscope aura dérivé au bout de quelques secondes, l'accéléromètre lui renverra la bonne valeur de rotation initiale.

Nous allons donc combiner ces données afin d'obtenir notre filtre :

$$angle = 0.98 * (angle + gyroData * dt) + 0.02 * accData$$

Les données du gyroscope sont intégrées à chaque impulsion de la carte avec la valeur précédente de l'angle. Ensuite, on les combines avec le filtre passe-bas de l'accéléromètre (grâce à la fonction atan2 -> accData = atan2(accData). Les constantes (0.98 et 0.02) doivent ajouter 1 mais elles peuvent être réglées de façon à faire correctement fonctionner le filtre.

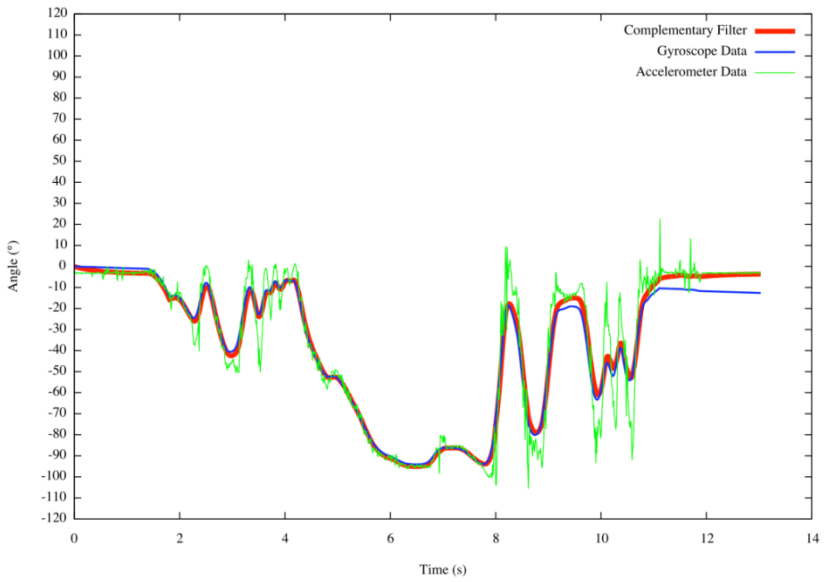


Figure 11 - Résultat du filtre trouvé sur le site de Peter-Jan.com

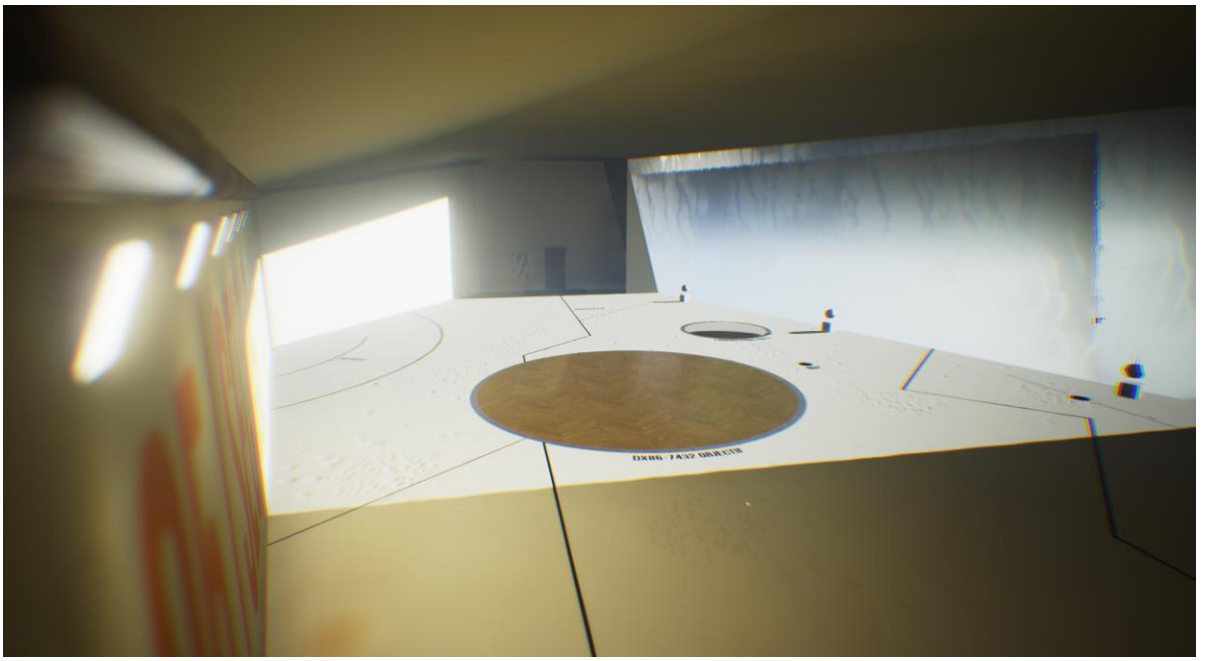
PARTIE 5 - ASSEMBLAGE ET CONCLUSION

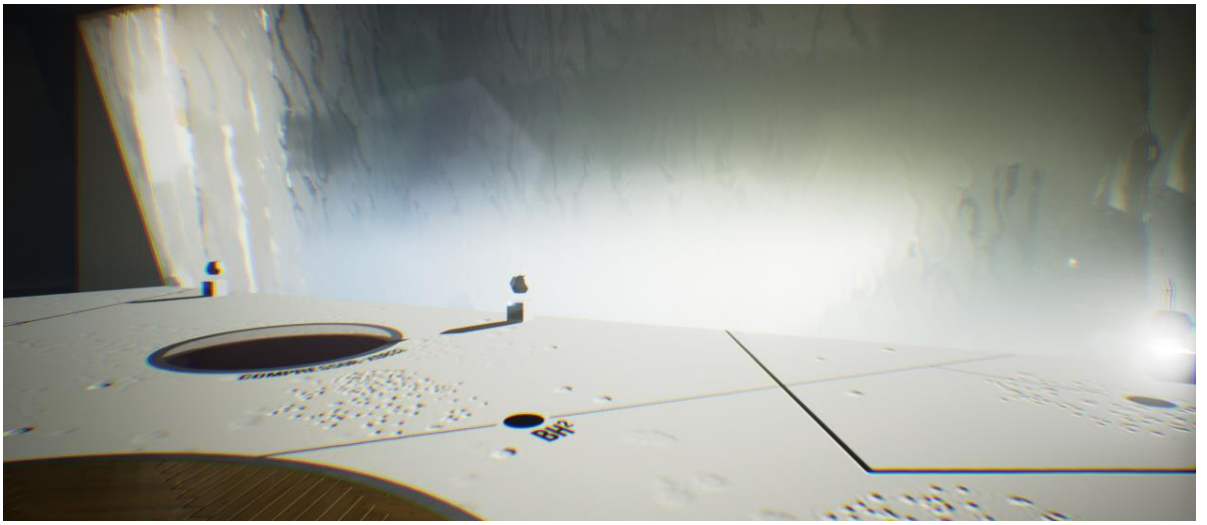
Une fois toutes les étapes réalisées, il nous reste à assembler tous les éléments afin d'obtenir notre casque.

Nous choisissons donc de créer un environnement graphique pour notre projet. Nous avons imaginé deux environnements dans lesquels le joueur pourra évoluer.

Figure 12 - Environnement 1 : Iles (Théau NICOLAS)

Figure 13 - Environnement 2 : Salle d'entraînement (Théau NICOLAS)





Steven LAFOLIE
Théau NICOLAS

Il n'y a pas de différence de programmation entre les deux environnements. Il s'agit juste de niveaux différents dans le même projet.

Pour que le jeu soit agréable, on rajoute quelques fonctionnalités pour la manette de jeu.

Nous avons acheté des châssis de casque de réalité virtuelle qui permettent d'y insérer n'importe quel portable. Nous avons la possibilité, une fois tous les détails de notre casque réglés, de faire imprimer en 3D ce châssis. Pour le moment cette solution nous convient pour les tests.

Résultats finaux

Le principe de base du casque est correct mais quelques problèmes persistent encore :

- La fluidité du jeu : les données de position possèdent déjà leur temps de latence ainsi que fonctionnement du casque. Lorsque le moteur de jeu est poussé à ses capacités optimales, le rendu à l'intérieur du casque est injouable. Ce problème provient de l'ordinateur avec lequel il fonctionne. Pour le faire tourner correctement, il est conseillé d'avoir une mémoire RAM d'au moins 8Go. Sinon, il est possible de baisser la qualité graphique du jeu afin d'obtenir une bonne fluidité.
- La qualité de l'écran : pour avoir une expérience de jeu correcte, il est conseillé d'utiliser un écran ayant une résolution d'au moins 1080p. Autrement, les détails seront plus difficilement visibles, ce qui nuit à la qualité.
- La longueur des fils : pour l'instant, les fils qui établissent la connexion ne nous permettent pas de jouer correctement. Le nombre de fils à brancher sur un ordinateur est aussi un problème pour l'utilisateur. Cependant, il est possible de réaliser le casque grâce à des réseaux sans fil mais beaucoup de problèmes surviennent ; la fluidité risque d'être affectée, la batterie du portable se décharge plus rapidement et la carte ARDUINO doit trouver une nouvelle source d'alimentation. Il nous faut donc utiliser des fils moins rigides et plus longs afin d'améliorer l'expérience.

CONCLUSION

Actuellement notre casque ne fonctionne pas de façon optimale car le filtre est difficile à appliquer. Nous sommes encore en phase de test, mais tous les éléments nécessaires à la réalisation du casque sont réunis. Il est donc possible de réaliser un casque de réalité virtuelle à moindre cout.

Au final ce projet a mobilisé beaucoup de temps de recherche mais aussi de compétences diverses dans le domaine de la programmation et de l'électronique. Il nous a appris à orienter nos recherches et à nous organiser dans notre travail. Ce projet a été pour nous une façon de concrétiser notre souhait d'obtenir un casque VR. Les casques actuellement en vente étant beaucoup trop chers, notre but était de rendre accessible cette technologie. C'est cette volonté qui nous a motivés tout au long de ce projet.

Au final, nous avons réalisé la partie électronique et informatique afin d'avoir le fonctionnement de base du casque. Les étapes suivantes sont les détails de programmation, le remplacement des éléments Arduino par des microcontrôleurs plus adaptés, puis le design du casque. Une fois ces étapes réalisées, il sera possible d'envisager le développement d'autres modules comme une manette ainsi que des jeux adaptés.

ANNEXES

Sommaire

ANNEXES 27

POSITION 27
AJOUT D'UNE MANETTE VIRTUELLE 27
REALISATION AVEC VISUAL BASIC ET COMPATIBILITE 27
BLENDER 28

Position

Dans notre cahier des charges, il était question d'agir sur la rotation mais aussi sur la position de notre casque. Le problème est que la position précise d'un appareil est difficile à obtenir. Il existe néanmoins des solutions comme des capteurs à placer à différents endroit (comme l'HTC Vive) ou même des cartes électroniques qui récupèrent la position au mètre près depuis satellite. La solution des capteurs paraît plus viable.

Ajout d'une manette virtuelle

Dans notre cahier des charges, nous devions simplement réaliser le fonctionnement du casque. Mais la finalité de notre projet serait d'intégrer au casque une manette de jeu personnalisée qui permettrait d'interagir directement dans le jeu comme dans le réalité. Cette manette peut prendre la forme souhaitée par l'utilisateur et une fois le principe du casque réalisé, il est très simple de le reprendre et de remplacer la caméra par notre manette. En décalant l'objet par rapport à son axe de rotation, il est possible d'obtenir l'illusion d'un objet tenu en main.

Réalisation avec Visual Basic et Compatibilité

Le problème de la programmation actuelle, est qu'elle n'est présente que sur le jeu vidéo développé uniquement à cet usage. Une bonne amélioration serait de le coder directement dans un programme extérieur afin de ne renvoyer que les informations nécessaires aux jeux vidéo. Cela nous permettrait d'obtenir une compatibilité complète. Une esquisse de ce programme à déjà été codée, mais le principal problème rencontré est la fausseté des informations récupérées sur le Port COM. Une fois ce problème réglé, il sera possible de réaliser plus de tests.

Blender

Le logiciel Blender est un logiciel de modélisation 3D Open Source permettant de créer et de réaliser des images de synthèse, des objets 3D, des animations ou des jeux vidéo. Il possède deux moteur de rendu (Render et Cycle) qui permettent de réaliser des images très réalistes pour certaines. Il possède aussi un moteur de jeu très intéressant car très simple à utiliser pour créer de petits jeux ou pour faire des maquettes de jeux sans coder une seule ligne.

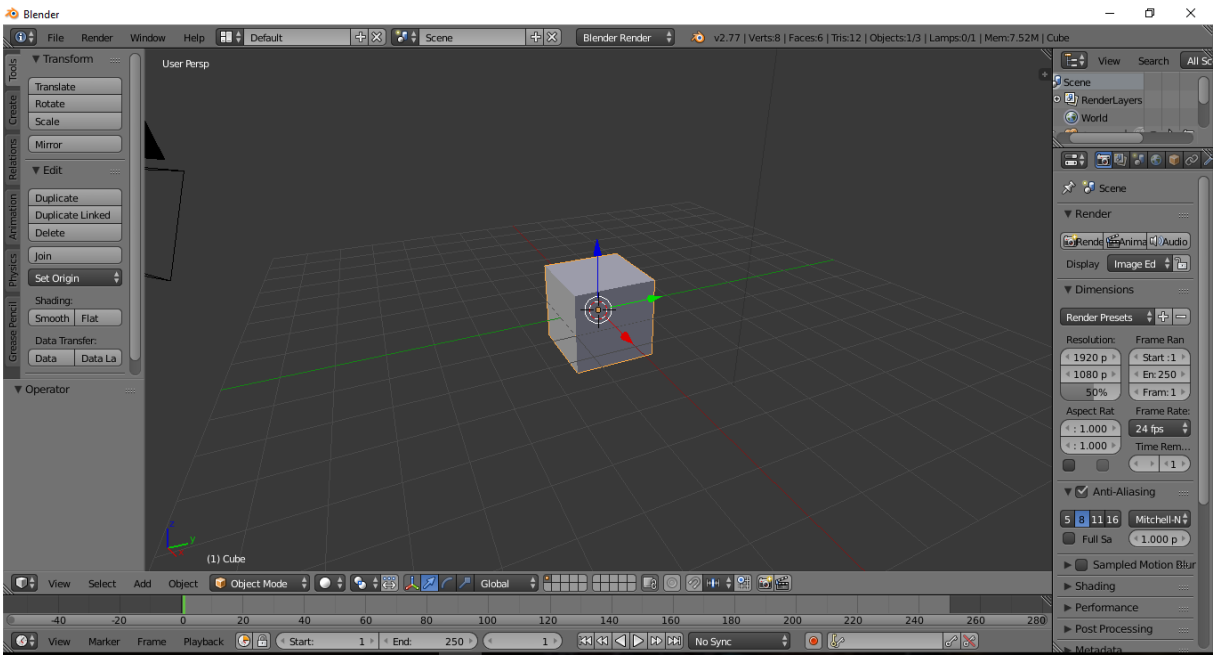


Figure 14 - interface principale de blender

L'avantage est que ces trois moteurs sont reliés directement au modélisateur ce qui donne une polyvalence impressionnante à Blender. Il est compatible Windows/Mac/Linux et peut réaliser des simulations de fluide, de vêtements et d'objets dynamiques très différents selon son utilisation.

Le moteur Render, moteur d'origine, permet de faire des rendus simples et très réalistes. Cependant il nécessite beaucoup de travail pour un rendu optimal puisqu'il a été créé dans le but de jongler entre de l'animation et du rendu d'image. Un rendu pouvant être très long à générer et la cadence d'image pour un film d'animation étant de 24 images/s minimum, il a été prévu pour une bonne vitesse d'exécution afin de réduire le temps de génération.

Le moteur Cycle, lui est conçu pour la performance. Il permet d'obtenir assez simplement un rendu impressionnant. Il est beaucoup plus lent dans ses calculs mais diminue considérablement le travail à effectuer pour le même résultat sur Render. Il peut aussi générer une animation. Cependant, même avec un ordinateur de bonne qualité, le temps d'exécution pour créer un film complet avec des graphismes corrects n'est pas satisfaisant**.** Il est possible de « calculer » le rendu sur des textures prédéfinies ce qui limite considérablement la consommation de mémoire vive de l'ordinateur, mais encore une fois, les rendus étant souvent complexes, le temps de calcul des textures est beaucoup trop long pour vraiment envisager un film complet.

Figure 15 - Rendu avec cycle réalisé par Théau NICOLAS

La réalisation d'objet 3D est basée sur un principe assez simple : on définit des points avec des coordonnées respectives que l'on relie ensemble afin d'obtenir une forme complète. La forme simple est appelée un mesh et il peut prendre la forme souhaitée par l'utilisateur, c'est l'outil ultime de création. De par ce principe, plus le mesh possède de points plus il est volumineux et plus l'ordinateur peine à l'afficher.

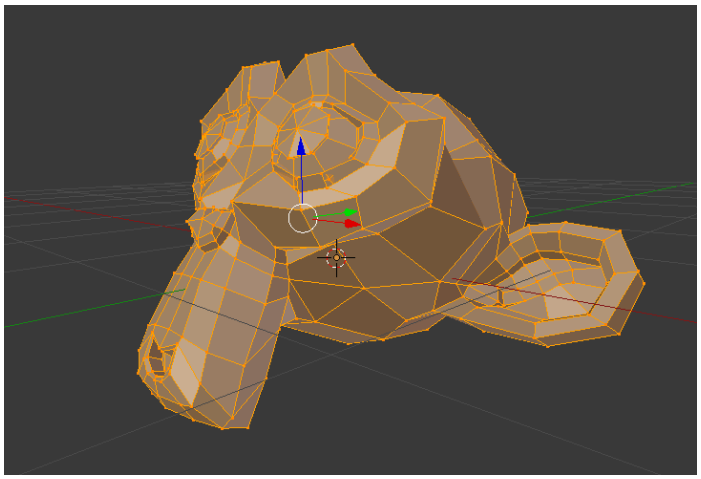


Figure 16 - Mesh simple – ici Suzanne la mascotte de Blender

Une fois la forme définie, on veut assigner à notre mesh une substance. Deux éléments la définissent :

- Le matériel : qui définit la matière de notre objet (bois, plastique, métal)
- Les textures : images qui définissent la couleur les variations de couleurs, tous les détails visibles

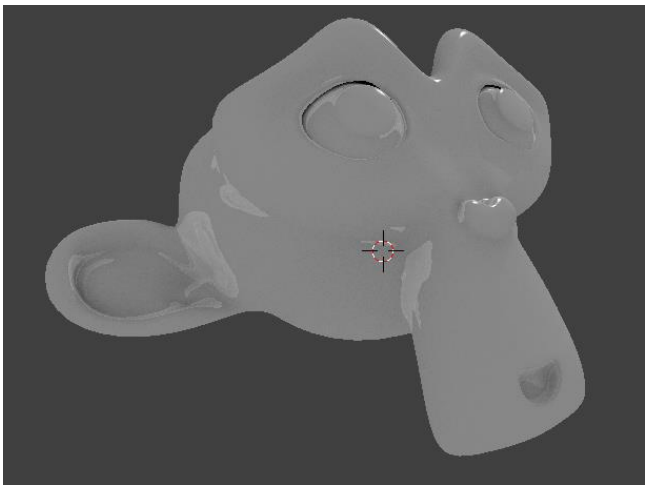


Figure 17 - de la céramique

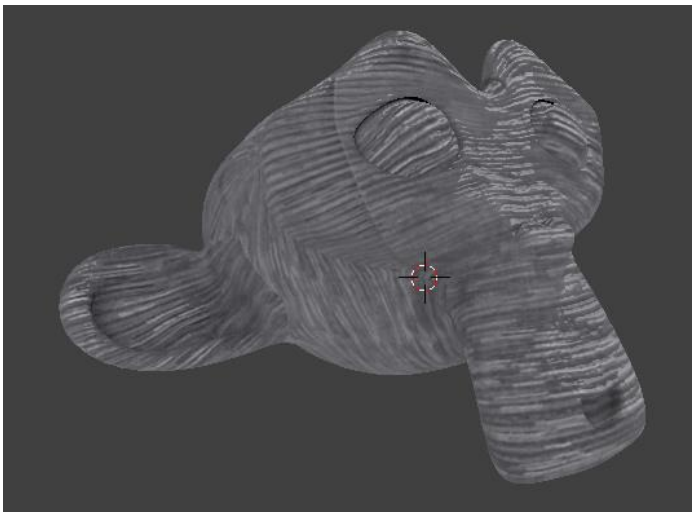


Figure 18 - de la pierre

Les textures ne peuvent pas être appliquées au hasard. Nous devons leur définir un plan à suivre selon lequel elles pourront se référencer. Ce plan est appelé UVMap il est défini sur deux axes, X et Y. Ce plan ne se définit pas seul. L'utilisateur doit « découper » le mesh de façon à obtenir un plan correct et compréhensible pour l'ordinateur :

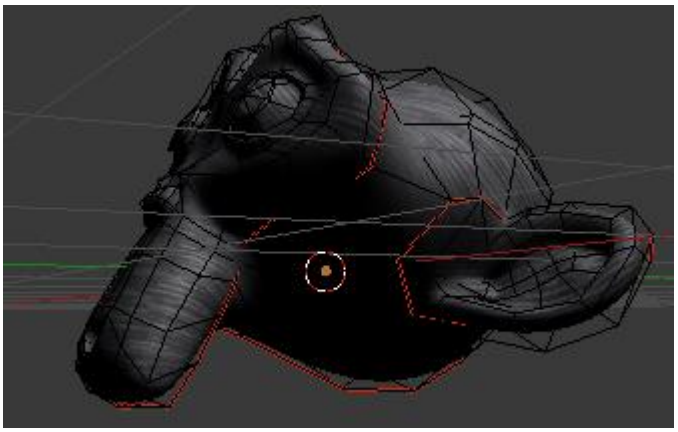


Figure 19 - Le découpage en rouge

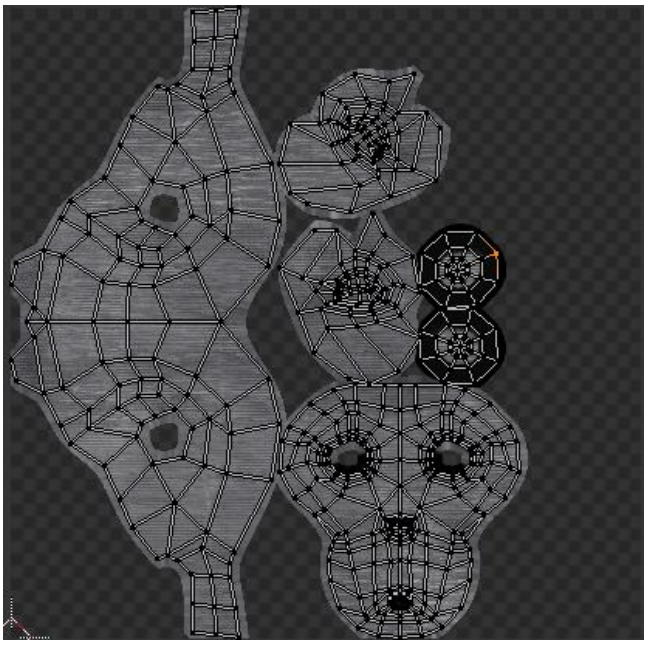


Figure 20 - Bon découpage

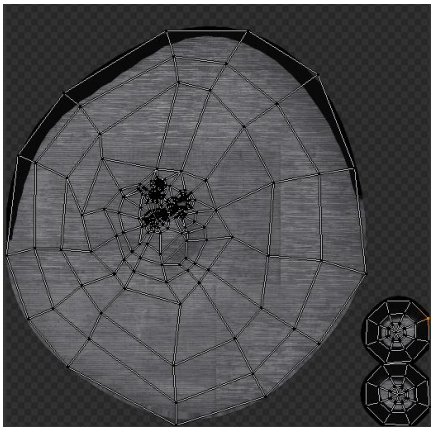


Figure 21 - Mauvais découpage

Ces étapes sont communes à tous les logiciels de modélisation 3D. Ce sont les principes fondamentaux de la création en 3 dimensions. Les matériaux et textures peuvent être combinés afin d'offrir le meilleur rendu possible. On les appelle des shaders. Ils sont codés dans le langage graphique (OpenGL ou DirectX) qui sera directement exécuté sur la carte graphique. Que ce soit en rendu image ou jeu vidéo, la qualité de ces shaders dépend surtout de la capacité du moteur et de la carte graphique pour leur vitesse d'exécution. Quand à leur qualité, ils demandent beaucoup de travail et connaissances à l'utilisateur.

La création d'un shader complexe est chronophage. En termes de jeu vidéo, un shader doit être convaincant mais aussi léger puisque le moteur est calibré pour faire tourner le jeu à 60fps ce demande d'être calculé extrêmement rapidement contrairement au rendu image. Pour améliorer la vitesse, on néglige souvent la qualité mais des techniques existent afin d'améliorer le rendu sans entamer la vitesse d'exécution.