ÉCOLE POLYTECHNIQUE DE MONTRÉAL

INF6802 Réalité virtuelle : Principes et Applications

Rapport de Projet : Vol à travers la ville

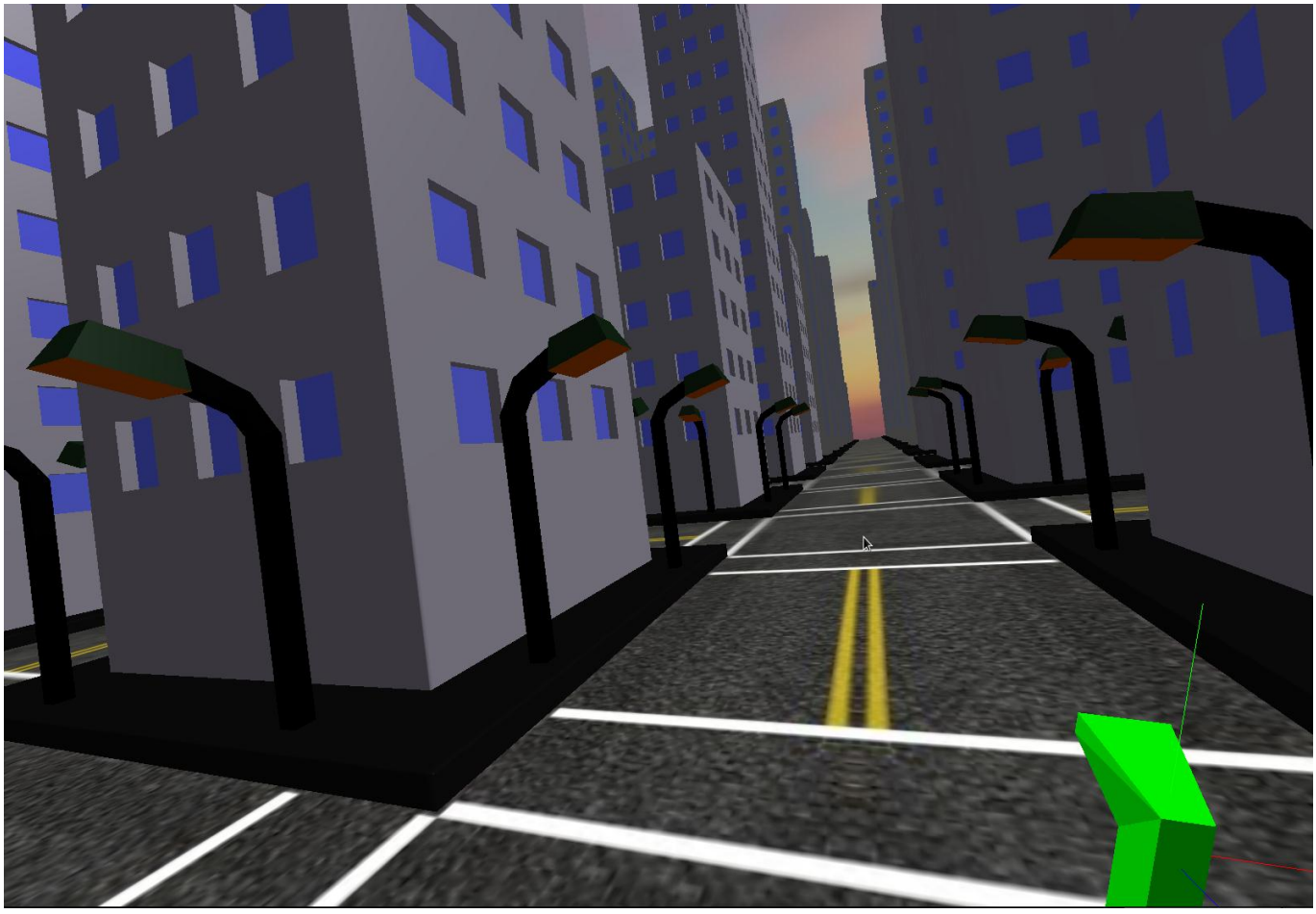
Maxime Hême 1597746

Nicolas Pellissier 1592908

Présentation

Notre projet intitulé *Vol à travers la ville* est un environnement de réalité virtuelle, présenté dans l'environnement interactif de la CAVE. Celui-ci porte sur la navigation de ce que l'on peut considérer être un vaisseau, ou moyen de transport libre de tout mouvement, à travers la représentation d'une ville, qui doit être assez ressemblante à une ville réelle actuelle.

L'objectif de notre projet était de rendre cette navigation le plus réaliste possible, afin que l'utilisateur puisse avoir la réelle impression de se trouver dans ce vaisseau, se retrouvant ainsi immergé dans un monde virtuel. Une grande partie de nos efforts a donc été dirigée par cet objectif d'une navigation fluide et intuitive.



Mode d'utilisation du programme

Dans ce paragraphe vont être présentées les possibilités de contrôle et d'interaction pour l'utilisateur, ainsi que quelques explications sur notre programme.

Tout d'abord, il n'existe qu'une unique façon de lancer l'application. Il n'existe aucune option qui pourrait changer le programme lors de son exécution.

Lors du lancement de notre programme, l'utilisateur se trouve au milieu d'une route, face à des bâtiments de différentes tailles, ce qui lui permet de se trouver au plein coeur de la ville dès le démarrage. Il est donc mis en situation au départ de l'application.

L'utilisateur se place alors au centre de la CAVE, muni des lunettes stéréoscopiques lui permettant de voir en 3D, et du wand, sorte de manette que l'on peut manipuler dans l'espace et muni de plusieurs boutons.

Une fois le programme lancé et l'utilisateur prêt à interagir avec l'environnement, voici ses possibilités :

- le bouton droit ("bouton 3") : celui-ci est un simple bouton poussoir qui permet l'arrêt et le démarrage du vaisseau. Si celui-ci est à l'arrêt, un appui permet de démarrer à une vitesse faible. Si celui-ci est en train de se déplacer, un appui sur ce bouton provoque l'arrêt du vaisseau. Le vaisseau peut se trouver à l'arrêt lors de deux situations différentes : soit l'utilisateur l'a choisi, soit il y a eu un phénomène de collision.

- la gâchette ("bouton 1") : celle-ci permet à l'utilisateur d'accélérer la vitesse du vaisseau lorsque celui-ci est en déplacement. Il faut la maintenir appuyée pour atteindre la vitesse maximale fixée, et le relâchement de cette gâchette décélère automatiquement la vitesse du vaisseau jusqu'à ce que celle-ci atteigne une vitesse minimale fixe.

- le bouton gauche ("bouton 2") : celui-ci ne permet cette fois pas d'interaction sur le vaisseau, mais sur l'environnement dans lequel évolue l'utilisateur. Un appui sur ce bouton permet en effet de modifier la skybox du monde virtuel ainsi que l'environnement sonore, qui sont tous les deux associés. Cela permet à l'utilisateur de choisir un environnement dans lequel il a envie de se déplacer. Six choix d'environnements différents sont proposés.

L'utilisateur pourra ainsi naviguer librement dans l'environnement. Il ne pourra cependant pas traverser les bâtiments ni le sol, pour lesquels des collisions apparaissent lorsqu'il se heurte à ceux-ci. La collision est un simple arrêt du vaisseau, il suffit donc à l'utilisateur de reprendre une direction opposée au bâtiment/sol et redémarrer le vaisseau grâce à l'appui sur le bouton droit.

Structures de données et algorithmes :

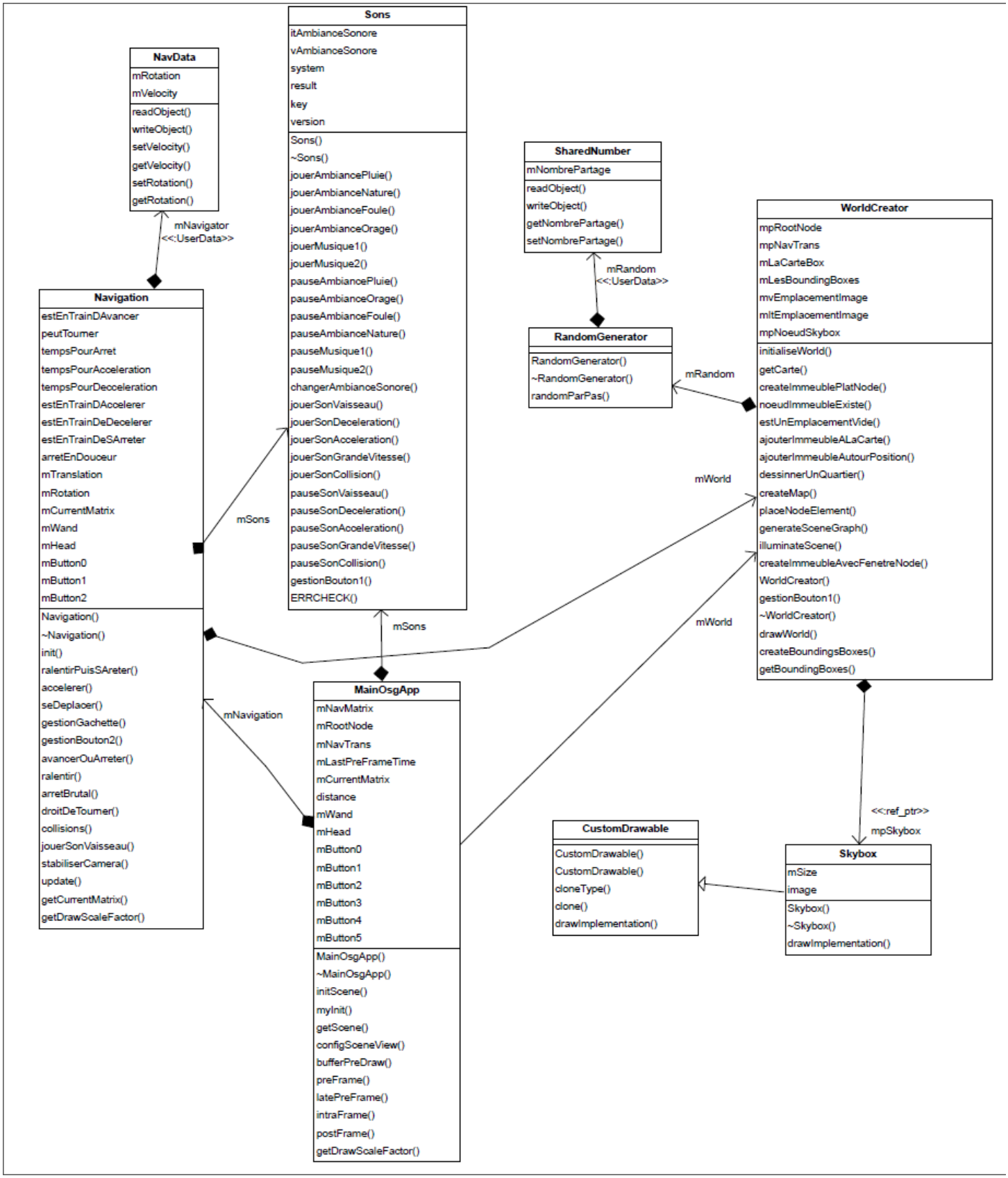
Nous avons séparé notre programme en neuf classes distinctes. L'une concerne les ambiances sonores ainsi que tous les bruitages concernant les déplacements et collisions du vaisseau. Elle contient la librairie FMOD. Une autre concerne la navigation du vaisseau, et contient donc les données telles que les rotations et translations du wand, les détections des appuis boutons, et des variables booléennes qui décrivent l'état du vaisseau pour aider à comprendre plus facilement son fonctionnement. Une autre concerne les graphismes de l'environnement ainsi que l'affichage des modèles 3d. Les autres classes sont des classes utilitaires qui comprennent un certain nombre de structures qui permet une utilisation optimisée des classes. Les différentes classes sont liées entre elles par des relations d'appartenance pour la plupart.

Nous avons aussi réalisé le partage des données entre chaque machine lors du lancement de l'application. En effet, le fichier *DonneesNavigation.h* permet de partager la vitesse de déplacement et la vitesse angulaire de rotation et le fichier *Util.h* partage la graine. Les machines ont donc toutes la même graine, ce qui permet que cette graine soit générée aléatoirement par une des machines et ensuite transmise afin de générer un environnement assez aléatoire.

Le fichier *WorldCreator.h* comprend une carte qui contient les immeubles en fonction de leur taille. Ils vont donc tous être générés (ou presque). Une autre carte pointe vers ces immeubles dont le rôle est de répartir tous les immeubles. Le même principe est utlisé pour la gestion des bounding box.

La classe *ImmeubleAvecFenetre* renvoie un noeud de LOD (*Level Of Details*) qui permet un affichage en fonction du niveau de détail. Il existe quatre niveau de détails différents : le premier affiche tous les détails, le second affiche tous les détails, sauf les lampadaires, le troisième affiche les immeubles avec moins de polygones, enfin le dernier n'affiche rien.

Des modifications de dernière minute ont permis une meilleure gestion des collisions : Il est maintenant plus aisé de redémarrer le vaisseau après que celui-ci ait été arrêté à cause d'une collision.


Sons
itAmbianceSonore
vAmbianceSonore
system
result
key
version
Sons()
~Sons()
jouerAmbiancePluie()
jouerAmbianceNature()
jouerAmbianceFoule()
jouerAmbianceOrage()
jouerMusique1()
jouerMusique2()
pauseAmbiancePluie()
pauseAmbianceOrage()
pauseAmbianceFoule()
pauseAmbianceNature()
pauseMusique1()
pauseMusique2()
changerAmbianceSonore()
jouerSonVaisseau()
jouerSonDeceleration()
jouerSonAcceleration()
jouerSonGrandeVitesse()
jouerSonCollision()
pauseSonVaisseau()
pauseSonDeceleration()
pauseSonAcceleration()
pauseSonGrandeVitesse()
pauseSonCollision()
gestionBouton1()
ERRCHECK()
NavData
mRotation
mVelocity
readObject()
writeObject()
setVelocity()
getVelocity()
setRotation()
getRotation()
mNavigator
<<:UserData>>
SharedNumber
mNombrePartage
readObject()
writeObject()
getNombrePartage()
setNombrePartage()
mRandom
<<:UserData>>
RandomGenerator
RandomGenerator()
~RandomGenerator()
randomParPas()
mRandom
WorldCreator
mpRootNode
mpNavTrans
mLaCarteBox
mLesBoundingBoxes
mvEmplacementImage
mItEmplacementImage
mpNoeudSkybox
initialiseWorld()
getCarte()
createImmeublePlatNode()
noeudImmeubleExiste()
estUnEmplacementVide()
ajouterImmeubleALaCarte()
ajouterImmeubleAutourPosition()
dessinnerUnQuartier()
createMap()
placeNodeElement()
generateSceneGraph()
illuminateScene()
createImmeubleAvecFenetreNode()
WorldCreator()
gestionBouton1()
~WorldCreator()
drawWorld()
createBoundingsBoxes()
getBoundingBoxes()
mWorld
mWorld
Navigation
estEnTrainDAvancer
peutTourner
tempsPourArret
tempsPourAcceleration
tempsPourDecceleration
estEnTrainDAccelerer
estEnTrainDeDecelerer
estEnTrainDeSArreter
arretEnDouceur
mTranslation
mRotation
mCurrentMatrix
mWand
mHead
mButton0
mButton1
mButton2
Navigation()
~Navigation()
init()
ralentirPuisSAreter()
accelerer()
seDeplacer()
gestionGachette()
gestionBouton2()
avancerOuArreter()
ralentir()
arretBrutal()
droitDeTourner()
collisions()
jouerSonVaisseau()
stabiliserCamera()
update()
getCurrentMatrix()
getDrawScaleFactor()
mSons
mSons
mNavigation
MainOsgApp
mNavMatrix
mRootNode
mNavTrans
mLastPreFrameTime
mCurrentMatrix
distance
mWand
mHead
mButton0
mButton1
mButton2
mButton3
mButton4
mButton5
MainOsgApp()
~MainOsgApp()
initScene()
myInit()
getScene()
configSceneView()
bufferPreDraw()
preFrame()
latePreFrame()
intraFrame()
postFrame()
getDrawScaleFactor()
<<:ref_ptr>>
mpSkybox
CustomDrawable
CustomDrawable()
CustomDrawable()
cloneType()
clone()
drawImplementation()
Skybox
mSize
image
Skybox()
~Skybox()
drawImplementation()

Discussions

Efficacité des méthodes utilisées :

La structure entière de notre programme a été pensée autour d'une efficacité maximale. Il a été privilégié des méthodes parfois courtes, mais qui évitaient des répétitions d'opérations, afin que le code en lui-même puisse être plus simple. Le nom des méthodes ainsi que celui des variables a été choisi de sorte qu'il soit plus facile de comprendre le code source. Il était donc plus facile pour nous-mêmes d'effectuer des modifications, et une tierce personne voulant modifier le code source devrait être en mesure de facilement le comprendre, aidé de plus par les commentaires présents tout au long du code.

Nous avons eu recours à l'utilisation d'un certain nombre de variables globales booléennes, qui nous ont aidé à distinguer certains états par leur facilité d'utilisation (*EstAlArret, EstEnTrainDeRalentir*…).

Seuls quelques problèmes de compréhension de certaines fonctions de la librairie FMOD ont donné lieu à un code source du fichier *Sons.cpp* certes organisé, mais peu optimisé avec de nombreux rappels des mêmes fonctions.

Choix des méthodes d'interaction avec l'utilisateur :

L'interaction de l'utilisateur sur le monde virtuel se fait à partir du wand, que l'on peut assimiler à une manette munie de trois boutons poussoirs (et un joystick non utilisable ici). Il a donc fallu faire les choix les plus judicieux concernant l'utilisation de ces 3 boutons.

Comme nous voulions faire de l'utilisateur le véritable pilote du vaisseau, nous avons choisi de dédier deux des trois boutons au pilotage direct du vaisseau.

Les principales interactions de l'utilisateur concernent donc les déplacements du vaisseau dans l'environnement. Comme vu précédemment, il peut alors arrêter et démarrer le vaisseau, accélérer et ralentir sa vitesse, et choisir la direction qu'il souhaite.

Les seules limites de déplacement sont imposées par des collisions avec des objets présents dans le monde, à savoir les bâtiments et le sol principalement. Cela signifie donc que l'utilisateur ne peut pas traverser ces objets.

En dehors de la ville, les limites sont imposées par la taille de l'environnement lui-même.

Afin de permettre un type d'interaction utilisateur / environnement autre que ceux qui concernent les déplacements, nous avons laissé le choix à l'utilisateur entre plusieurs environnements différents. Il lui suffit en effet de presser un bouton pour faire défiler six environnements différents. Ces changements consistent à changer la skybox (environnement visuel) et l'environnement sonore associé à la skybox en question.

Difficultés rencontrées :

La principale difficulté à laquelle nous avons été confrontés est l'implémentation de l'algorithme de navigation. C'était aussi notre priorité sur l'ensemble du projet.

Cette difficulté est venue du fait que puisque nous laissions l'utilisateur entièrement libre de ses mouvements, il pouvait faire des rotations selon les 3 axes de l'environnement. Cela posait le problème que lorsqu'il faisait une rotation autour de l'axe vertical et autour d'un axe horizontal simultanément, il ne se retrouvait pas parallèle au sol une fois ce mouvement terminé. Cela n'était donc pas assez intuitif car il lui aurait fallu effectuer d'autres manoeuvres pour rétablir cette situation, et il aurait été quelque peu perdu en faisant face à cela.

Nous avons dans un premier temps pensé à limiter le nombre de rotation à une seule rotation possible à la fois, ce qui aurait éviter cette situation inconfortable. Seulement cela allait à l'encontre de ce qui avait été décidé au départ, à savoir une totale liberté des mouvements.

Nous avons alors choisi d'implémenter un algorithme de stabilisation du vaisseau lorsque celui-ci ne serait pas parallèle au sol, et qui rétablirait donc le vaisseau de façon assez discrète pour que ce mouvement ne perturbe pas l'utilisateur et qu'il n'ait pas l'impression que ce mouvement soit désagréable et vu comme une intervention du programme contre sa volonté.

Cette solution que nous avons adoptée n'est pas encore parfaite puisque le mouvement a parfois lieu de façon différente selon la direction du vaisseau, et peut parfois laisser apparaître quelques à-coups. Elle est cependant fonctionnelle dans la plupart des cas, et assez discrète pour ne pas être désagréable à l'utilisateur.

Nous avons aussi fait face à un autre problème, celui de l'exportation de modèles 3d. Nous avons créé nos modèles à l'aide du logiciel Blender, mais leur exportation s'est révélée assez problématique. Nous avons tout d'abord essayé d'exporter nos modèles au format 3ds, mais Blender ne permet pas une exportation correcte des textures avec ce format, les fichiers au format 3ds étant mal gérés avec les shaders. Nous avons alors essayé de les exporter au format .obj, sans plus de succès car même si ceux-ci exportent correctement les textures, ils supportent mal les lumières, ainsi que les shaders (qui enlèvent les textures). Il a alors fallu éviter de mettre des textures lorsque l'on voulait utiliser des shaders, Nous avons finalement décidé d'exporter nos modèles au format osg à partir de Blender.

Nos shaders ont alors fonctionné correctement. On remplace les textures par des matériaux avec les couleurs. Les shaders s'appliquent enfin avec succès. Le résultat final est donc correct, même si les calculs nécessitent plus de temps pour se faire. Pour pallier à cette perte de rendement, nous avons joué sur le niveau de détail : lorsque l'on s'éloigne ce ne sont plus des modèles osg mais obj qui contiennent des textures mais plus d'effet de lumière.

Avantages de l'environnement :

La priorité de notre travail étant la navigation, celle-ci est l'un des plus grand avantage de l'environnement. Nous avons ainsi fait en sorte de la rendre la plus intuitive possible à l'utilisateur, ce qui donne une impression d'immersion plus grande.

Cela est d'autant plus vrai que lors d'une accélération du ciel vers le sol de la ville, une sensation physique de vitesse est éprouvée lorsque l'on se rapproche des bâtiments de la ville.

Un autre avantage de notre environnement se trouve dans la génération de la ville en elle-même. Elle est en effet générée de façon aléatoire, ce qui a pour avantage de changer l'environnement à chaque lancement de celui-ci. Même si nous ne l'appliquons que pour changer la taille des bâtiments (les bâtiments gardent le même emplacement), cette gestion de l'affichage partiellement aléatoire pourrait être utilisée pour d'autres fonctions, comme par exemple la génération aléatoire de l'emplacement des bâtiments, de la forme de ceux-ci, ou encore pour générer d'autres objets comme des voitures.

Désavantages de l'environnement :

L'environnement est limité dans l'espace, et atteint ses limites avec le nombre de bâtiments générés. Il sera donc impossible de générer une très grande ville. Ceci sera surtout vrai si l'on change la forme des bâtiments (de simples rectangles) en faisant apparaître un plus grand niveau de détails, ou d'autres types d'installations dans la ville (terrains de foot, piscines, parcs…).

Les détails graphiques sont donc assez limités si l'on veut avoir une très grande taille (celle que nous avons maintenant restant acceptable).

Un autre désavantage est que l'on ne peut pas vraiment voir le vaisseau dans lequel nous sommes en train de nous déplacer. Il aurait été d'un niveau de réalisme supérieur de voir les panneaux de contrôle du vaisseau pour nous mettre en position de "pilote" de l'appareil, et ainsi avoir l'impression de vraiment piloter ce vaisseau.

Améliorations possibles :

Avec plus de temps alloué au projet, nous aurions pu effectuer quelques améliorations à notre programme :

- Niveau de détails de l'environnement plus élevé : plus de détails sur les bâtiments, bâtiments de formes différentes, voitures, foule en mouvement...

- Meilleure définition des limites de la ville : lorsque l'on arrive aux limites de la ville, ne pas donner l'impression que celle-ci est un carré flottant dans le vide. On pourrait par exemple créé une mer artificielle autour de la ville qui donnerait l'impression que celle-ci se trouve sur une île, ou encore utiliser un paysage de ville qui donnerait l'impression d'une ville infinie.

- Gestion plus réaliste des collisions : en effet, lorsque le vaisseau arrive à toute vitesse sur un immeuble, celui-ci ne fait seulement que s'arrêter. Il serait plus réaliste que celui-ci subisse un effet de rebond et/ou qu'il y ait une dégradation visible sur l'immeuble.

- Création d'un tableau de bord qui donnerait à l'utilisateur une plus forte impression de pilotage du vaisseau.

- Utilisation de sons plus réaliste pour les bruitages du vaisseau ainsi que l'environnement sonore.

Utilisations possibles de l'environnement :

L'application la plus probable de notre environnement serait pour un jeu vidéo, dans lequel on pourrait se déplacer grâce à un véhicule de type vaisseau qui aurait les mêmes libertés de déplacement que celui que nous avons créé.

Étant donné les possibilités de navigation, il pourrait aussi être imaginé que notre programme soit utilisé dans un environnement réel reconstitué pour pouvoir se déplacer d'un point à un autre avec facilité et fluidité.

On pourra trouver tous les codes sources de notre programme dans le dossier :
/reseau/bonne/exports/users/heme/RealiteVirtuelle/inf6802_realite_virtuelle/trunk/ProjetFinal