Stardust

Rapport troisième soutenance

Koin-Koin z'Team

CRESTEY Benoît
CALVET Bruno
CARRIGNON David
EGGENSPIELER Rémi

INFOSUP

23 avril 2003

Table des matières

1 Introduction

Notre groupe est toujours composé de Benoît Crestey (SUP B2) le chef de groupe, David Carrignon (SUP A2), Rémi Eggenspieller (SUP D2), Bruno Calvet (SUP B1).

Nous développons toujours un jeu de simulation spatiale en 3D qui a connu des best-sellers tels que Wing Commander, X-Wing Vs Tie Fighter et le très célèbre Starlancer, à noter que depuis la dernière soutenance nous avons découvert un nouveau jeu dans ce genre : Freelancer, duquel nous nous sommes inspirés pour certains petits détails.

1.1 Répartition des tâches

Mê si nous avons beaucoup travaillés ensemble, voici une courte liste des principales tâches de chacun pour cette soutenance :

Nom	**Répartition des tâches**
CALVET Bruno	– Détection des collisions. – Gestion des dégats. – Gestion des effets sonores et de la musique. – Site Internet.
CARRIGNON David	– Gestion de la souris. – Rendu des lasers. – Formules mathématiques. – Texturage. – Modélisation.
CRESTEY Benoît	– Missiles Guidés. – Intelligence Artificielle. – Gestion des vaisseaux. – Gestion des lasers. – Travaux dans l'interface.
EGGENSPIELLER Rémi	– Gestion du vertex buffer dynamique. – Gestion des explosions. – Affichage des explosions. – Site Internet.

2 Avancement du projet

2.1 Formules Mathématiques

Comme le monde de notre jeu est en 3D, nous avons besoin de faire de nombreux calculs mathématiques tels que des changements de repères entre autres utilisés pour l'Intelligence Artificielle.

2.1.1 Calcul de la matrice des Particules

Pour afficher les particules nous devons les tourner en direction de la caméra pour qu'elles paraissent sphériques. Pour celà nous calculons une matrice orientant les objets face à la caméra. Celle-ci n'est calculée qu'une seule fois par boucle d'affichage, et nous permet d'afficher toutes les particules du jeu, car il suffit de changer trois de ses valeurs pour déplacer le rendu de la particule dans le repère monde. Cette méthode réduit ainsi les calculs du processeur, dans le but d'obtenir un jeu plus fluide.

2.1.2 Changement de repère d'un vecteur

Le calcul de changement de repère est très important dans notre jeu. Il nous permet de déterminer d'où doivent partir les lasers et les missiles. Le moyen qui nous parut le plus simple pour calculer les changements de repère, fut d'utiliser les matrices de direct3D. Nous avons donc créé deux fonctions : l'une permettant de déterminer dans une base quelconque de R^3 les coordonnées d'un vecteur exprimé dans la base canonique de R^3 et l'autre permettant de passer des coordonnées d'une base quelconque dans une base canonique.

2.2 Rendu Laser

Tout jeu dans l'espace qui se respecte doit avoir de jolis lasers qui traversent l'écran. Il fallut donc trouver un moyen d'en créer. Nous avons d'abord recherché sur internet comment réaliser graphiquement des objets énergétiques comme dans les derniers jeux commerciaux. Mais malheureusement, les tutoriaux trouvés étaient basés sur des sytèmes de particule classiques. Donc, pour créer un laser suivant leurs méthodes, il fallait positionner une dizaine de particules décalées les une par rapport aux autres. L'effet aurait été très esthétique, mais pour afficher dix lasers, il aurait fallut afficher une centaine de particules : ce qui n'est pas acceptable au niveau des performances ; puisque notre moteur devrait rapidement être obligé d'afficher au moins une centaine de lasers.

Nous avons donc dù trouver une autre méthode. Nous avons alors longuement observé des jeux tels que *Starlancer* ou *Freelancer*.

2.2.1 Les differentes méthodes de rendu

Dans certains de ces produits multimédias,nous avons effectué plusieurs rotations de caméras autour de ces lasers afin de déceler les forces et faiblesses de deux principes de rendu de lasers. La première méthode consiste à créer une image des lasers de côté et d'en afficher trois, decalés de soixante degrés autour du vecteur direction du laser. Ce système est très facile à programmer en direct3D, et très joli lorsque les lasers sont vus de côté mais lorsqu'il sont vu de faces ou de dos l'effet n'est pas du tout réaliste.

Contrairement à la première méthode, la seconde reste réaliste indépendamment de la direction du laser par rapport à la caméra. Ces lasers sont composés d'une particule et d'une trainée. La trainée doit tourner autour du vecteur direction du laser de telle manière que sa normale

pointe vers la caméra. Celle-ci paraîtra ainsi cylindrique ou cônique suivant la texture choisie. Ce second sytème de rendu des lasers fut choisit par le groupe.

2.2.2 Réalisation du rendu

La méthode retenue ne fut pas la plus simple, car nous avons dù trouver le moyen de faire tourner la trainée vers la caméra par nous même.

Pour cela, nous avons dù étudier comment fonctionnent les matrices de Direct3D. Car au lieu d'utiliser les fonctions prédéfinies pour calculer les matrices, nous calculons directement les vecteurs X,Y,Z de la matrice ainsi que ses coefficients de translations. Les coefficents sont très simple à déterminer, c'est simplement la position du laser. Pour les trois vecteurs celà se complique : on prend d'abord la direction du laser, sa position et celle de la caméra. On commence par calculer le vecteur partant du laser et pointant vers la caméra. Ensuite on fait le produit vectoriel de ce vecteur avec celui de la direction du laser. Ce qui nous donne un vecteur perpendiculaire au plan de ces deux vecteurs. Puis on fait un nouveau produit scalaire entre ce vecteur et la direction du laser. Les vecteurs X,Y,Z de la matrice sont alors la direction du laser et les deux vecteurs calculés par les produits scalaires. Il suffit de les normaliser et de remplir la matrice.

Ensuite il ne reste plus qu'à afficher une particule à la position du laser à l'aide de la matrice des particule qui est calculée une seule fois par boucle d'affichage et qui retient la rotation des particules pour que celles-ci soient tournées vers la caméra.

2.3 Modélisation 3D

Pour cette soutenance, nous avons modélisé un second vaisseau à l 'aide du logiciel 3D studio Max. Comme pour le premier vaisseau, ce logiciel fut utilisé pour pouvoir facilement importer notre nouvel objet 3D. La méthode utilisée fut la modélisation à l'aide de lignes, dans le soucis d'optimiser le nombre de polygones. Grâce à cette technique, la forme du modèle possède une qualité esthétique très satisfaisante en comparaison de son très faible nombre de polygones (aux environs de 250). Le vaisseau modélisé devait être de design extra-terrestre. La plus grosse difficulté fut de donner à ce vaisseau son aspect. Sa forme y est pour beaucoup mais sa texture y contribue aussi. Nous avons donc choisi une image tirée d'une serie appelée Babylone 5. Celle-ci fut retravaillée et ensuite appliquée sur le vaisseau. Nous avons ainsi réussi à réaliser un second vaisseau avec un design d'une certaine qualité. Ce qui permet de jouer avec un chasseur humain ou extraterrestre aux designs totalement différents.

2.4 Collisions

2.4.1 Récapitulatif

Pour la deuxième soutenance nous avions déjà présenté une ébauche des collisions et fait une démonstration de celles-ci. Les collisions dans notre jeu étaient gérées par des Axis-Aligned Bounding Box. Tout d'abord les éléments étaient triés en fonction de leur coordonnées d'abscisses puis une fois le tri fait il y avait des tests de collisions entre chaque élément seulement une fois et si ils collisionaient ils étaient détruits. Ceci était bien sur pas du tout réaliste et nous ne l'avions fait qu'à des fins de démonstration nous comptions bien sur gérer des dommages en fonction de collisions entre vaisseaux et entre vaisseaux et armes aussi.

2.4.2 Travail effectué

Pour cette troisième soutenance nous avions prévue différente choses :

- Simplifier dans la mesure du possible les algorithmes existant
- Définir des sous-Bounding Spheres pour affiner les collisions
- Eventuellement utiliser des OBB (oriented Bounding Box)

Nous avons légèrement simplifié l'algorithme de collisions en y intégrant la notion de rayon maximum, en effet lors du tri on recherche l'élément ayant le rayon le plus grand et on le stocke. Ensuite lors des test de collisions on utilise ce rayon pour éviter certains tests qui seraient inutiles. L'algorithme de tri n'a pas été simplifié car il est le plus simple possible.

Comme il était prévu nous avons mis en place l'utilisation de sous-Bounding Sphere pour affiner les tests et les rendre graphiquement plus précis.

Pour ce faire nous avons décider de gérer les sous-sphères dans un arbre binaire contenant à chaque noeud les coordonnées de la sous-sphères par rapport à l'origine du vaisseau, le rayon de la sous-sphère et deux pointeurs, un vers la sous-sphère droite et un autre vers celle de gauche. Toutes ces informations sont stockées dans un fichier que nous avons créé nous même qui es lu au lancement du jeu et toutes les informations sont stockées au lancement du jeu dans une liste chaînée. Pour reconnaitre facilement les fils d'un noeud de l'arbre nous avons utilisé la notation utilisée en cours d'algo, le noeud initial est nommé 'A' son fils gauche '0' et le fils droit '1' et pour accéder à leur fils droit on rajoute '1' à la suite de leur nom et '0' pour accéder à leur fils gauche, si on tombe sur un noeud non existant on met comme pointeur nil ainsi on construit un arbre binaire facile d'utilisation et contenant tout ce qu'il nous faut pour utiliser les sous-sphères.

Nous avons apporté quelques modifications à la fonction principale de calcul de collisions, nous utilisons maintenant seulement des sphères pour les collisions et plus des AABB. En effet nous avons décidé d'utiliser uniquement des Bounding Sphere afin que ce soient toujours les même calculs qui soient fait et aussi pour des raison de clarté dans le code. Pour faire les test de collisions grâce aux sphères on calcul le carré de la longueur entre le centre des deux éléments $(carre(X1-X2)+carre(Y1-Y2)+carre(Z1-Z2))$ ainsi que le carré de la somme des rayons $(carre(R1+R2))$, on utilise des carré pour éviter de calculer des racines carrés qui sont assez gourmandes en temps de calcul, puis on compare ces deux résultats, si le carré de la distance est inférieur au carré de la somme des rayon alors il y a collision. Si il y a collisions entre les deux sphères principales des éléments on appelle un autre fonction qui fera les tests entre les sous-sphères. On a décidé de ne pas faire de test entre missile et laser ainsi qu'entre laser et laser

car ceci est assez ridicule à nos yeux qui plus est cela économise des tests.

Cette fonction prend en paramètre les pointeurs sur les 2 éléments se collisionant, elle stocke leurs coordonnées et ensuite il y a plusieur cas de figure, une collision entre deux vaisseauxs de l'ordinateur, entre un vaisseau de l'ordinateur et un missile et entre un vaisseau de l'ordinateur et un laser. On test ensuite où se situe la collision, toujours par rapport au vaisseau (un seul des deux s'il s'agit d'une collision vaisseau-vaisseau),si celle-ci est à gauche du centre du vaisseau ou à sa droite on appelle ensuite la fonction permettant de faire les tests sur les sous-sphères en indiquant les sous-sphères qu'il faut tester, il faut noter que pour les missiles et les lasers nous n'utilisons pas de sous-sphères car nous avons jugé que cela était inutile car nous utilisons des sphères assez petites pour ces éléments là.

La fonction appellée va chercher dans la liste chaînée les coordonées de la ou les sous-sphères passées en paramètres, et récupère les coordonnées ainsi que le rayon, on utilise ensuite une fonction permettant de replacer dans les coordonnées du monde les sous-sphères dont les coordonnées sont définies par rapport au vaisseau, elles sont ainsi correctement positionnées dans le monde et on refait des tests sur la distance des éléments et le rayon des sphères, on reteste ensuite la position des sous-sphères pour appeler la fonction de manière récursive sur les autres sous-sphères, une fois terminé tous les appels de cette fonction elle renvoit un booleén qui est à true si il y a eu collision et qui est à false sinon. Ce résultat permet ensuite de déclencher les évènements ayant lieu en cas de collision entre vaisseaux ou avec les armes.

Pour la soutenance précédente nous n'avions pas encore fait de collisions pour le vaisseau du joueur et nous l'avons donc fait pour cette fois-ci. Nous avons fait une fonction qui parcourt simplement la liste de collision du jeu, en omettant les laser et les missiles, et qui fait le même test de collision que pour les vaisseaux de l'ordinateur mais l'opération faite si il y a collision est différente en effet nous avons implémenté une façon plus réaliste de représenter les collisions entre vaisseaux.

Lorsque deux vaisseaux se collisionent, que ce soient des vaisseaux de l'ordinateur ou le vaisseau du joueur et un de l'ordinateur, il se passe une réaction que nous avons voulu assez réaliste, les deux vaisseaux se repoussent. Pour ce faire ce que nous avons fait est assez simple, nous avons rajouté dans la définition des vaisseaux deux paramètres : le vecteur de déviation et la vitesse de déviation, ainsi lorsque deux vaisseaux se collisionent chacun prend le vecteur direction de l'autre comme son vecteur de déviation et la vitesse de l'autre comme sa vitesse de déviation, qui plus est leur vitesse respective est modifiée en fonction de la norme de la somme des vecteur sur la somme des normes des vecteurs, $(norme(V1 + V2)/norme(V1) + norme(V2))$, si leurs vecteurs direction sont paralèlles leur vitesse ne sera pas modifiée sinon elle sera diminuée en raison du choc, la vitesse de déviation diminue au cours du temps afin que les vaisseaux ne dérivent pas indéfiniment et il leur est aussi infligé des dégats au bouclier ou à la coque si le bouclier est déchargé.

Pour l'IA nous avons décidé de créer une fonction permettant de savoir si un endroit précis était accessible. Cette fonction est comme toute les fonctions de collisions concernant le test de collisions, on y met les paramètres suivant : coordonnées (x,y,z) et le rayon de l'endroit à tester, si l'endroit est accessible la fonction renvoit true et si il y a eu collision elle renvoit false montrant ainsi que l'endroit est inacessible.

2.4.3 Bilan sur les collisions

Nous pouvons donc considérer que le travail sur les collisions est terminé. En effet ce que nous comptions faire a été fait et les collisions en utilisant les Bounding Sphere fonctionne correctement. Ceci va nous permettre de nous consacrer à d'autre parties importantes du projet.

2.5 Gestion du jeu

2.5.1 Gestion des vaisseaux

Les différents vaisseaux présents sont toujours gérés dans une liste chainée, qui est parcourue deux fois : une fois pour la gestion, une fois pour l'affichage. vu que la création dans la mémoire d'un vaisseau devenait de plus en plus gourmande (il fallait stocker toujours plus de valeurs de positions, d'armes, de données pour l'intelligence artificielle,...), pour éviter de stocker inutilement des valeurs pour chaque vaisseau, nous avons créé des enregistrements contenant les valeurs pour chaque modèle de vaisseau. Ainsi lors de la création d'un nouveau vaisseau dans la mémoire, on ne stocke pas toutes les informations le concernant mais les informations individuelles, plus une variable entière qui permet d'accéder aux informations que les vaisseaux ont en commun.

Parmis les informations communes, on trouve par exemple les définitions de sphères nécessaires aux collisions, les points de vie, les résistances, les positions des différentes armes ainsi que leur type (lui-même un enregistrement des propriétés des différentes armes, vu que plusieurs vaisseaux peuvent avoir des armes en communs), ... Ainsi, à chaque fois que l'on a besoin d'une information commune, on compare le type du vaisseau, puis on va rechercher les valeurs correspondantes dans le bon enregistrement.

Les posisions et caractéristique de chaque Laser, Missiles ou Lumière présent sur la coque de chaque vaisseau est géré de la même facon, évitant de stocker toutes ces valeurs à chaque fois.

Un autre avantage de cette méthode est que la modification des valeurs est beaucoup plus simple : si par exemple on veut changer la fréquence de tir d'un type d'arme, cette modification affectera tous les vaisseaux l'utilisant.

Nous avons ajouté de nombreuses données afin de gérer les points de vie de la coque et du bouclier de chaque vaisseau, de gérer l'énergie (chaque arme possède à présent des fréquences de tirs, des dépenses en énergie, des portées), qu'il faut mettre à jour à chaque cycle de jeu, toutes ces valeurs interviennent plusieurs possibilités allant de l'utilisation ou non pour chaque arme, jusqu'à détruire le vaisseau.

2.5.2 Gestion des lasers

Dès que l'affichage des lasers à été finalisé, il a fallu mettre en oeuvre un moyen de les gérer. Nous avons donc créé une liste chainée spécialement pour cela (nous avions au début pensé inclure les lasers avec les missiles, mais leur type presentant peu de similitudes, nous avons préféré utiliser deux listes). Les calculs qu'entrainent les lasers ont été ajoutés : chaque laser posséde un type qui le défini, et ainsi on sait quels sont ses dégats suivant s'il touche le bouclier ou la coque, ses fréquences de répétition, le cout d'énergie pour son utilisation (l'energie est régénérée suivant chaque type de vaisseau, et incite le joueur à ne pas garder toujours la touche de tir), sa vitesse, sa portée,...

2.5.3 Gestion des missiles

Ainsi que détaillé dans la partie Intelligence artificielle, nous avons mainteant des missiles simples, c'est à dire qui avancent sans changer de direction, et des missiles guidés, qui une fois leur cible acquise, sont capables de la suivre suivant ses déplacement. Nous avons également ajouté

un temps de vie à chacuns de ceux-ci, permettant leur disparition au bout d'un temps donné. Bien évidemment, et comme tous les calculs de temps dans notre jeu, si ceux-ci sont calculés plus ou moins suivant la vitesse de résolution de la boucle principale du jeu, nous prenons en compte le temps de résolution et adaptons en conséquence la valeur de temps de l'objet.

2.6 Intelligence Artificielle

Un des grands travaux prévus pour cette soutenance fut de commencer la gestion d'une intelligence artificielle capable de piloter un vaisseau, et de se battre contre le joueur de façon à simuler un comportement humain.

Pour commencer, la première étape fut de rendre les vaisseaux capables de rejoindre une position prédéfinie dans l'espace, de manière autonome.

2.6.1 Calcul de trajectoires

Pour rejoindre un point dans l'espace, le déplacement du vaisseau devait ressembler le plus possible au déplacement d'un joueur humain, d'autant plus que le vaisseau piloté par l'ordinateur possède exactement les mêmes caracteristiques que celui d'un joueur. Étant donné que chaque vaisseau possède une rotation maximale définie d'avance, cela l'empêche d'adopter des angles de trajectoire trop vifs, mais ces caractéristiques doivent être prises en compte lors du déplacement (ralentir pour tourner sur une plus courte distance par exemple).

2.6.2 Principe

Afin de trouver la direction à adopter, la première étape est de calculer la position de la cible à rejoindre, en fonction de la position et la direction de l'objet à diriger. Une fois que l'on a réussi à calculer ce vecteur (on utilise la position et le quaternion d'un premier objet ainsi que la position d'un second dans l'espace et on obtient un vecteur de direction par rapport au premier objet), il est trés facile de comparer avec la direction actuelle et de corriger la trajectoire et la vitesse.

2.6.3 Première application : missiles guidés

La première utilisation de ce calcul de trajectoire dans le jeu fut utilisée pour la gestion des missiles, qui fut un bon moyen de vérifier que les calculs étaient valides.

Nous avons créé des missiles capables de suivre un vaisseau préalablement séléctionné, qui doivent être capables de suivre une cible en mouvement. Le missile possède donc un pointeur sur sa cible, ainsi il peut connaitre sa position en temps réél. Bien sûr, comme celui-ci n'est pas toujours aussi rapide qu'un vaisseau, et que son coefficient de rotation est limité suivant ses caractéristiques, il peut corriger sa trajectoire en fonction de celle de sa cible. Egalement il ne fallait pas oublier de vérifier qu'aucun pointeur de cible ne pointe sur un vaisseau qui est détruit, car celui-ci pointerait vers son ancienne adresse mémoire qui n'est plus utilisée, on risquerait alors d'obtenir une erreur de pointage et risquer le plantage de l'application. On prend donc soin de parcourir les objets qui sont succebtibles de pointer vers un vaisseau avant sa destruction en mémoire.

Bien entendu, si l'on ne défini pas de cible, les missiles gardent leur trajectoire de départ.

2.6.4 Application Recherchée : Pilotage de vaisseaux

Le but recherché de ce guidage était bien évidemment de créer une intelligence artificielle qui soit capable de combattre. Pour celà, nous avons commencé par établir différentes phases avec des réactions différentes :

- Une phase de fuite, dans laquelle le vaisseau atteint une distance nécessaire afin d'attaquer l'ennemi, il prend pour celà sa vitesse maximum.

- Une phase de mise en position, qui permet au vaisseau de se mettre face à sa cible. Pour celà il adopte une vitesse basse, afin de tourner plus rapidement.

- Une phase d'attaque dans laquelle le vaisseau, dès qu'il est à une distance assez proche (toutes les armes ont une portée définie), adopte une vitesse moyenne, et commence à tirer dès que sa cible se trouve dans un certain angle de tir.

- Une phase de déviation : suivant sa vitesse, le vaisseau, dès qu'il détecte une future collision (avec une fonction associée au moteur de collisions, on teste la présence d'objets ou de vaisseaux dans des sphères réparties autour de lui), choisit une nouvelle direction de déviation afin d'éviter la collision. Cette nouvelle direction peut être calculée aléatoirement pour paraitre plus réaliste.

Bien entendu, toutes ces actions s'enchaînent et s'enclenchent suivant la distance et la position du vaisseau et de sa cible.

On utilise donc beaucoup les calculs de distances et les transformations de vecteurs (pour rapporter une position en fonction du vaisseau), on utilise aussi les caractéristiques de l'intelligence artificielle : en effet les vitesses de fuite, mise en position ou attaque sont stockées, ainsi que les distances correspondantes à chacune des phases. Pour ajouter plus de réalisme, ces caractéristiques sont calculées aléatoirement (on prends en fait des valeurs prédéfinies que l'on modifie aléatoiremement) lors de l'activation de l'IA : ainsi, chaque vaisseau possède des caractéristiques différentes qui permettent de ne pas avoir les mêmes réactions.

Par la suite nous allons tenter d'ajouter des tactiques spéciales pour éviter les armes adverses, la gestion des équipe ou de changement de cible automatique. Il faudrait aussi faire des tactiques permettant de faire varier la difficulté pour le joueur.

2.7 Interface du jeu

2.7.1 Affichage de textures, résolutions

Pour le moment, nous n'avons pas encore créé de système permettant de changer la résolution en cours de jeu, mais nous utilisons des différentes résolutions pour le développement (vu que nous travaillions tous sur des tailles d'écrans différentes). Mis à part l'initialisation Direct3D, le problème fut d'adapter l'affichage des objets en 2D appliqués à l'écran, en effet, la taille et la position de ceux-ci variaient suivant la résolution de l'écran, et comme nous ne voulions pas nous limiter à une seule résolution, nous avons créé des fonctions afin d'afficher toujours avec la même proportionnalité. Comme le redimmensionnement des textures coute un peu, nous avons décider d'adopter une taille unitaire qui est le 1024*768, le redimmensionnement des images intervenant que lorsque des résolutions supérieures sont utilisées (accessoirement inférieures, mais le 800*600 est de plus en plus rare, nottament pour les joueurs).

Une fois ces fonctions d'affichage 2D réalisées, nous avons put être en mesure d'afficher plusieurs indicateurs sur l'écran. On peut également noter que ces procédures gèrent complétement l'affichage des images transparentes (nous sommes à présent capables de gérer les degrés de transparence sur les images). Nous avons donc commencé par changer l'affichage de la précédente console, qui ne nous paraissait pas très jolie, nous avons pu afficher un pointeur autre que celui de windows, ainsi que des indicateurs de jeu : barres d'énergie, de bouclier, de vie.

2.7.2 Affichage de la liste des vaisseaux à proximité

Pour commencer à implémenter les informations affichées dans le cockpit, nous avons créé un affichage des vaisseaux à proximité, dans certaines limites de distance, avec différenciation de la cible, qui apparait d'une autre couleur que la liste.

2.7.3 Sélection de cibles

Nous avons également créé un système permettant de sélectionner une cible : grâce à l'appui d'une touche, on peut sélectionner un vaisseau, ce qui permet au joueur de mieux le viser, de l'avoir mieux en évidence, et surtout d'être capable de bloquer la cible pour lancer un missile guidé. La cible est désignée en mémoire grâce à un pointeur, ce qui nous permet d'obtenir ses informations (position), d'ainsi calculer la position du vaisseau ciblé par rapport au vaisseau du joueur, et ainsi être en mesure de calculer la position du viseur sur l'écran. Bien sur, nous avons pris garde à bien changer le pointeur si la cible visée est détruite : le pointeur passe au suivant. A noter que nous avons ajouté à la cible des barres de vie et de bouclier, afin de permettre au joueur de connaitre les dégats qu'il a affligé à son adversaire.

2.7.4 Sélection des armes

Nous avons également ajouté un système proposant au joueur de sélectionenr les armes avec lesquelles il fera feu, et lui permettre de les désactiver, par exemple pour économiser son énergie plus longtemps et favoriser le temps pendant lequel il pourra tirer plutot que sa puissance d'attaque. Le système est le même pour les missiles, et il faut noter que les vaisseaux joués par l'ordinateur possèdent le même système, ce qui nous permettera de donner la psosibilité au système d'intelligence artificielle de sélectionner ses armes.

2.8 Gestion de la souris

Pour cette soutenance, nous avons inclus la gestion de la souris à l'aide de DirectInput. Contrairement au clavier, ce périphérique n'est pas géré avec un buffer, car il ralentit la réaction. Ce choix fut décidé car nous utilisons la souris dans le déplacement du vaisseau du joueur, cele-ci doit donc réagir plus rapidement comme dans un first person shooter. Pour se déplacer avec la souris, nous utilisons sa position à l'écran contrairement aux Doom-Like qui utilisent l'intensité du mouvement à un instant donné. Par exemple, lorsque la souris est dans le coin haut gauche de l'écran, le vaisseau lève le nez et tourne à gauche. Alors que si la souris est au milieu de l'écran, le vaisseau ne tourne pas. Le déplacement à la souris est très intéressant car il est analogique et il devient beaucoup plus facile de viser un vaisseau enemi.

La gestion actuelle de ce périphérique est aussi très intéressante pour la conception future du menu, car nous retenons les coordonnés de la souris sur l'écran.

2.9 Création d'un *vertex buffer* dynamique

Le vertex buffer est la structure clé dans le dessin de primitives en 3D. Il est alors important de bien le gérer. Mais lorsque l'on veut modifier les données contenues, à savoir : les coordonnées des vertex, les normales, et les coordonnées de texture ; celà devient impossible. Or, pour l'affichage des particules, que ce soit l'affichage des explosions, les lasers, les lumières du vaisseau, il est nécessaire de pouvoir en modifier les données des vertex. Ainsi, un vertex buffer dynamique fut créé.
Le vertex buffer dynamique est une structure statique (tableau) contenant un nombre défini de vertex. Son utilisation n'est nécessaire qu'au moment d'afficher la particule. On peut donc passer la main au vertex buffer normal après l'affichage de la particule.

2.10 Gestion et affichage des explosions

Lors de la dernière soutenance, l'affichage du texte fut grandement amélioré. Pour cette troisième soutenance, l'affichage et la gestion des explosions devaient être réalisés.
Il va sans dire que dans un tel jeu, les explosions se doivent d'être esthétiquement réussies afin d'ajouter une dose de réalisme. En effet, lorsqu'un ennemi est détruit, il serait dommmage que l'explosion se limite à une texture statique appliquée à un carré. Il fut alors décidé que celles-ci soient composées de plusieurs frames animées, puis d'une onde de choc, améliorant drastiquement la sensation d'immersion. Les textures utilisées ont soit été utilisées comme telles, soit quelque peu modifiées.
Bien que simple en apparence, la création d'explosions plus poussées graphiquement n'a pas été une simple affaire, ce que nous verrons ci-après.

2.10.1 Principe

Les textures se représentent sous la forme d'images de 512 pixels de côté. Sept textures sont utilisées. Chacune de ces texture comporte les différentes étapes de l'explosion nécéssaires à son animation. Comme les frames d'une textures ont même hauteur et largeur, il était plus aisé d'écrire un algorithme d'extraction du carré de texture nécessaire, contrairement aux textures de polices dont les frames n'étaient pas régulières, nécessitant ainsi de relever chaque coordonnée de texture et de les placer dans un tableau.
Il suffit alors de déterminer le nombre de frames contenues dans une ligne pour déterminer la largeur d'une frame dans la texture. Ceci fait, il est alors facile d'afficher la frame désirée.
La frame désirée pouvant être affichée, l'animation devient alors possible. En créant un compteur s'incrémentant de 1 à chaque boucle, et grâce aux opérateurs "div" et "mod" l'explosion s'anime

alors.
Il faut permette de placer cette animation près du vaisseau désiré ; ceci fait il ne reste alors plus qu'à composer l'explosion avec plusieurs animations de différentes textures selon les goûts de l'auteur, et d'y ajouter l'onde de choc finale si populaire.

2.10.2 Réalisation

Le principe mis en place, il fallut alors écrire l'algorithme d'affichage d'explosions. Celui-ci ne nécessite de connaître que le nombre de frames contenue sur une ligne. Celles-ci étant déclarées en constantes, l'algorithme peut alors s'en servir pour déterminer la largeur en pixels d'une frame (512 divisé par la constante). Une fonction retournant la largeur de le frame fut créée.

Un problème se posa alors. Il fallait, pour l'affichage des explosions, pouvoir modifier librement les coordonnées de texture des vertex. L'accès direct au vertex-buffer étant impossible, mais il fut décidé de créer un vertex-buffer dynamique, représenté sous la forme d'un tableau de quatres vertex. Celui-ci sera utilisé spécialement pour les explosion, nécessitant un changement de coordonnées de textures permanent. Lorsque la frame est affichée, le vertex buffer dynamique cède alors la place au vertex buffer principal pour l'affichage des autres objets.

La procédure d'affichage de la frame étant mise en place, un autre problème se posa. En effet, cette procédure prend en paramètre la texture voulue, et la frame désirée représentée par un entier. Or, pour pouvoir incrémenter la frame le compteur doit impérativement être global à la boucle de rendu. Il fut alors préféré de créer une liste chaînée d'explosions. Lorsqu'un vaisseau explose, un élément est alors ajouté dans la liste. L'élément est initialisé avec un entier qui représente sa durée de vie fixée à 60. Tant qu'il reste des éléments dans la liste, une procédure décrémente alors de 1 la durée de vie de l'explosion. L'entier désignant le numéro de frame est alors calculé en soustrayant à 60 la durée de vie actuelle. Quand la durée de vie est nulle, l'élément est alors enlevé de la liste. On procède alors à l'affichage de chaque explosion contenue dans celle-ci.

L'étape suivante fut de placer la frame sur le vaisseau. Ceci se résume à stocker le vecteur coordonnées (TD3DXVECTOR3) dans un champ de la liste chainée. On effectue alors une translation de la matrice monde selon ce vecteur. Mais pour obtenir un peu plus de réalisme, placer toutes les particules à la même coordonnée n'est pas l'idéal. Sept vecteur sont alors créés aléatoirement dans un certain ordre de grandeur. Comme chaque explosion est constituée de sept sous-explosions, on effectue une addition entre chaque nouveau vecteur et le vecteur du vaisseau. On obtient ainsi à chaque vaisseau détruit, une explosion originale.

Il ne restait plus alors qu'à créer l'onde de choc vingt frames avant la fin de la durée de vie de l'explosion. L'onde de choc est un carré texturé dont les coordonnées de vertex sont proportionnelles à la frame en cours. Le carré s'agrandit ainsi au fil du temps jusqu'à la fin.

Ainsi, lorsqu'un ennemi est abattu, une magnifique explosion s'ensuit, pour notre plus grand bonheur.

2.10.3 Impressions

Loin d'être évidente de prime-abord, la gestion et surtout l'affichage des explosions se révélèrent compliqués à mettre en oeuvre. Celà a nécessité de comprendre mieux la philosophie de direct3D,

ses fonctions. Ainsi, malgré les difficultés, voir une explosion très esthétique et animée lors de la disparition d'un ennemi, nous procura une grande satisfaction.

2.11 Sons

2.11.1 Introduction

Dans tout les jeux existant sur le marché il y a de la musique, celle-ci participe à l'ambiance du jeu, une musique douce sera plutôt reposante alors qu'une musique plus aggressive excitera plus le joueur le poussant à être plus actif, une musique pourra aussi retranscrire la situation actuelle, glauque, combat, énigme, etc... Qui plus est les jeux sont parfois identifiable par les musiques qui leur sont spécifique, la musique du menu de deus-ex est assez connus par nombre de joueurs par exemple. Les effets sonores sont aussi une part importante de tout jeu, ils apportent une assez grande dose à l'interactivité car lorsqu'un joueur fera une action non seulement verra-t-il les effets mais il entendra aussi ceux-ci ce qui est en général très agréable, les effets permettent aussi de casser la monotonie d'un jeu en prévenant le joueur des évènements. Il était donc nécessaire que nous intégrions du son à notre jeu que ce soit de la musique ou des effets sonores.
Plusieurs possibilités s'offraient à nous :

- DirectSound
- Fmod

Nous avons choisi d'utiliser Fmod pour la très simple raison que c'est très rapide à mettre en place, en effet nous voulions pouvoir jouer des musiques et des sons de la façon la plus simple et rapide possible et Fmod est très pratique pour cela.

2.11.2 Gestion du son

Nous avons tout d'abord choisi quelques effets sonores tiré du jeu Freelancer qui est un jeu de simulation spatiale en 3D sortit récemment, y ayant joué quelque peu nous avons choisi quelques sons comme des tirs de lasers, de missiles et d'explosions. N'ayant pas les moyens de créer les sons et les musiques nous les avons reprise d'un jeu et c'est ce que nous comptons faire pour les autres sons que nous aurons à intégrer. Nous avons choisi quelques musiques d'ambiances qui pour le moment ne collent pas forcément à l'action du jeu mais qui sont là pour masquer le vide créé par l'abscence de musique. L'adaptation des musiques en fonction de l'action du jeu pourra éventuellement se faire plus tard.

Le fonctionnement de Fmod est assez simple, au démarrage le jeu charge les musiques et les sons et stocke les pointeurs vers ceux-ci dans des tableaux, un pour les effets et un pour la musique, ensuite si l'on veut jouer un effet précis on lance une procédure avec comme paramètre le numéro de l'effet et celui-ci sera joué, pour les musiques c'est différend, la procédure pour jouer une musique utilise une variable globale pour dire quelle musique il faut jouer, ainsi quand la procédure est appelée on joue la musique suivante. Nous avons aussi créé une petite procédure permettant de voir si une chanson est finie, si c'est le cas on joue la musique suivante.

2.11.3 Prévision pour le son

Nous avons donc mis en place correctement l'utilisation de musique et d'effets sonores ce qu'il nous reste à faire est de mettre à chaque action un son correspondant et trouver de bonne musiques d'ambiance.

3 Avance du projet

Tâches prévues dans le cahier des charges :

Moteur graphique avec vue du cockpit et affichage des ennemis.

Le moteur graphique possède toutes ces caractéristiques ainsi que beaucoup d'autres, l'affichage des missiles, des lasers et d'un bonne partie de l'interface est déjà réalisée.

Début de la gestion de l'intelligence artificielle

L'intelligence artificielle est déjà capable de combattre à peu près crédiblement, pour l'améliorer il va falloir lui apprendre à éviter toutes sortes d'objets, et implémenter d'autres techniques d'attaques pour diversifier.

Début de la gestion des armes

Les lasers et les missiles (guidés ou non) sont gérés dans le jeu, tout ce que nous pourrons faire pour les améliorer sera d'ajouter de nouvelles textures et caractéristiques à ceux-ci.

Gestion du script d'événements

Le script d'événements est totalelement opérationnel et de nouvelles fonctionnalités lui sont ajoutées de jours en jours.

Démonstration des collisions

Les collisions ont encore été améliorées, elle seront présentées lors de la soutenance.

Début de la modélisation / texturage

Un nouveau vaisseau complétement créé par nos soins vient d'être ajouté à la collection d'objets de notre jeu.

Présentation de l'avancement du site Internet

Le site Internet est toujours opérationnel et en fonction.

Tâches réalisées non prévues dans le cahier des charges :

Affichages des enenmis

Nous n'avions pas prévu d'être déjà capables d'obtenir une explosion assez réaliste. Nous voulons continuer dans ce domaine afind de pouvoir créer une effet de fragmentation de différents blocs du vaisseau.

Gestion de la souris

Nous ne comptions pas gérer l'utilisation de la souris dans le jeu, mis à part pour le menu, cependant, on peut à présent se diriger gràce à celle-ci, selectionner ses armes ou d'autres vaisseaux.

Intelligence Artificielle

Nous pensions juste créer un algorithme de déplacement et nous avons déjà une bonne partie de l'intelligence artificielle qui est gérée.

Lasers, missiles

Les missiles guidés sont déjà totallement gérés, les lasers aussi, les seuls travaux à finir sont de l'ordre du texturage et de changements de données pour un plus grand gameplay.

Collisions

Gràca aux fonctions de calculs en 3D, les collisions sont plus précises, on espère qu'elles le seront encore plus lors de la dernière soutenance

Interface

L'interface est plus avancée que prévue, certaines fonctionnalités comme la gestion de cibles avec la souris ne devaient pas être mises en oeuvre

Gestion du Son

Nous ne pensions pas être capables de fournir un environnement sonore pour cette troisième soutenance, de nombreux effets sont inclus ainsi qu'une musique d'ambience.

4 Ressources

Différents sites Internet visités :

- *http ://www.gamasutra.com/features/20000330/bobic_01.htm*, site très complet concernant les collisions bien que ceratins article soient assez technique
- *http ://www.gamedev.net/reference/articles/article1234.asp*, site intéressant détaillant pas mal d'opérations mathématiques pour les collisions utilisant les Bounding Shperes.
- *http ://www.developer.com/tech/print.php*, site expliquant les procédures de DirectInput our la souris.
- *http ://www.jedi-project.org*
- *http ://www.delphisource.com*
- *http ://www.game-dev.net*, site contenant divers tutoriaux sur l'utilisation des quaternions
- *L'intro DirectX* aux éditions CampusPress, livre de DirectX intéressant et assez clair
- *DirectX Delphi Game Programming*, Difficile d'accès, car en anglais, mais très clair et pratique pour une initiation à DirectX
- *L'Intro DirectX* - Campus Press - Difficile d'accès car les exemples sont en C++, mais traite de parties très intéressantes au niveau de Direct3D
- *Delphi 4 guide du développeur* - Campus Press - Tès pratique par l'initiation à l'environnement Delphi
- *Turbo Pascal 7* - Eyrolles - Bien expliqué, pratique pour se former au langage Pascal

5 Travaux prévus pour la soutenance finale

Collisions

Nous comptons continuer le travail sur les collisions en simplifiant le plus possible les algorithmes existant et en affinant la précision de la détection.

Gestion du Son

Nous pensons peut être inclure une ébauche de la gestion du son qui nous donnera plus d'interactivité à notre jeu.

Début de la gestion de l'IA

Nous avons prévu d'intégrer une IA à notre jeu et nous vous montrerons surement une ébauche de celle-ci.

Moteur graphique

Pour la troisième soutenance nous présenterons un menu assez simple.

Affichage des armes

Nous allons créer des nouvelles armes notamment l'affichage de laser réaliste.

Site Internet

Nous continuerons bien sur d'améliorer le site Internet.

6 Conclusion

Nous avons complété tous les travaux pour cette troisième soutenance, et nous avons conservé notre avance sur le cahier des charges. La soutenance finale approche à grand pas. Nous avons l'impression d'avoir été efficaces dans notre organisation. En outre, notre projet s'étoffe de plus en plus, et est devenu jouable, pour notre plus grand bonheur.

7 Annexes

Vue d'une formation de vaisseaux.

Nouveau modèle de vaisseau.

Nouvelle console.

Explosions sur un vaisseau.