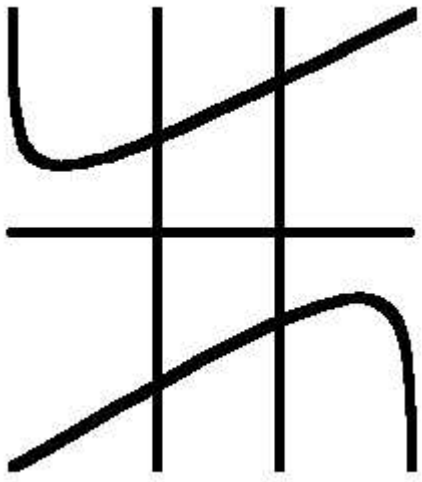
Info-SUP : E2

SpeedWay

Rapport de soutenance finale



Grignon Joël : grigno_j
Meermann Priscillien : meerma_p
Quadrat Quentin : quadra_q
Rozovas Roman : rozova_r

2001-2002

Table des matières

Chapitre 1

Introduction

1.1 Présentation du projet :

Nous allons vous présenter un jeu de voitures où la bagarre est la seule issue pour gagner. En effet, ce projet mélange le quake-like et les courses de voitures.
Je retiens votre attention pour vous dire que le principe du jeu diffère des autres projets que vous auriez pu voir jusqu'à maintenant (enfin, à ma connaissance). On ne va pas jusqu'à prétendre que nous allons révolutionner le monde du jeu mais nous avons trouvé ce principe assez sympa. C'est pourquoi nous vous conseillons de lire le manuel d'utilisation du jeu avant de vous battre avec votre ennemi. Pour ceux qui sont intéressés par la mise en oeuvre de ce programme, nous avons là toute les théories mises en place dans notre projet.

1.2 Scénario :

Après une pénurie mondiale des stocks de production de carburants, une guerre éclata entre les pays, et la course au carburant commença. Une large partie de la population a été décimée et seuls quelques êtres débrouillards ont réussi à survivre. L'essence étant de plus en plus rare, des combats éclatèrent sur les routes pour s'approprier l'or noir. Votre mission, si vous l'acceptez, est de survivre dans cette jungle mécanique. N'oubliez pas que les amis sont rares et que des bandes barbares circulent, tuent et pillent.
Une bande en particulier sévit dans ce désert aride. Leurs membres sadiques et pervers se font connaître sous le nom de Hatila et imposent leur volonté par la force et la torture. Personne n'a encore réussi à sortir indemne d'une rencontre avec eux. Le seul moyen de survivre est de convertir les unités énnemis pour que seul votre puissance règne sur la territoire.

1.3 Comment y jouer :

SpeedWay est un jeu 3D où la camera suit le véhicule du joueur en permanence.
Le but est de survivre tout en se faisant des amis. Le jeu se fera en équipe d'un maximum de cinq joueurs. Il y aura trois équipes, les bleu, les neutres et les rouges. Sur la carte on trouvera des bonus representer sous forme transparente contenant des munitions et des bonus : lance-missiles, mines, mitraillettes, sniffer.
Si le niveau d'essence ou de niveau de vie est à zéro, alors la voiture s'arrètera et seuls le premier venu la sauvera mais cela aura pour effet de la convertir dans l'équipe de la voiture sauveteuse. L'essence est rare par consequent il faut posseder le plus d'amis pour avoir le plus d'escence. Encore une petite chose, lorsque vous convertisser une voiture usagè, vous lui donnez la moitier de son escence.

1.4 Répartion des Tâches

	Grignon	Meermann	Quadrat
Moteur 3D	*	*	
Modélisation	*	*	*
Son			*
Contrôle Clavier	*	*	
IA	*		*
HUD		*	
Effet speciaux		*	*
Moteur évenementiel	*	*	*
Interface de démarrage	*	*	
Moteur Physique	*		
Site Web	*		

1.4.1 Répartition pour la première soutenance

1. **Grignon :**
   (a) recherche de sources exemples sur internet ou sur des livres.
   (b) apprendre à programmer en delphi.
   (c) début de lecture du Red Book (pour OpenGL).
2. **Meermann :**
   (a) recherche de sources exemples sur internet ou sur des livres.
   (b) apprendre à programmer en delphi.
   (c) début de lecture du Red Book (pour OpenGL).
3. **Quadrat :**
   (a) recherche de sources exemples sur internet ou sur des livres.
   (b) apprendre à programmer en delphi.
   (c) à apris de se servir de MNOGL et de GLScene (OpenGL pour delphi).
4. **Roman :**
   (a) recherche de sources exemples sur internet ou sur des livres.
   (b) apprendre à programmer en delphi.
   (c) ébauche du site web.

1.4.2 Répartition pour la deuxième soutenance

1. **Grignon :**
   (a) création d'un programme d'implémentation des fichiers 3Ds.
   (b) recherche de documentation sur les collisions.
2. **Meermann :**
   (a) création d'un programme d'implémentation des fichiers 3Ds.
   (b) reprise de l'initialisation de l'affichage.
3. **Quadrat :**
   (a) commencement de l'IA.
   (b) création d'une map et d'une bagnole avec 3Ds.
4. **Roman :**
   (a) mise à jour du site web.

1.4.3 Répartition pour la troisième soutenance

1. **Grignon :**
   (a) création d'une map et d'un véhicule avec 3Ds.
   (b) configuration au démarage.
2. **Meermann :**
   (a) création d'une map et d'un véhicule avec 3Ds.
   (b) création des armes et gestion des evenements qui y sont liés.
   (c) création des bonus et gestion des evenements qui y sont liés.
   (d) mise à jour du site web.
3. **Quadrat :**
   (a) poursuite de l'IA.
   (b) gestion du son.

1.4.4 Répartition pour la soutenance finale

1. **Grignon :**
   (a) finalisation de l'IA.
   (b) gestion des collisions camera joueur.
   (c) débugage.
2. **Meermann :**

(a) programme d'installation.

(b) mise à jour du site web.

(c) débugage.

3. **Quadrat :**

(a) création d'une pelleteuse avec 3Ds.

(b) finalisation de l'IA.

(c) débugage.

Chapitre 2

Site web

Le site web a été réalisé sous Microsoft Frontpage pour une plus grande facilité et rapidité de création. Pour les graphismes du site, ils ont été réalisés avec Photoshop 5 et Paintshop 7.

2.1 Utilité du site

Le site web fut réalisé afin de montrer notre projet au reste du monde et ainsi nous faire valoir. C'est aussi une passerelle de communication. En effet nous permettons aux autres personnes de nous envoyer des idées. D'ailleurs cela nous a bien servi. Le site fut assez visité ; nous tenons d'ailleurs à en remercier les visiteurs. Nous n'avons malheureusement pas mis de compteur pour les visites. Mais les mails que l'on a reçus nous ont quand même prouvé qu'il avait une certaine fréquentation.

2.2 Location

Voici l'addresse où est hébergé le site.

`http://perso.wanadoo.fr/priscilien`

2.3 Disposition et rubriques du site

Pour l'apparence du site nous avons décidé de faire simple. Les menus sont situés dans une colonne à gauche et l'on peut accéder aux principales pages du site de cette façon. Cette colonne est fixe donc l'internaute s'y retrouvera facilement et ne recherchera pas l'emplacement d'un lien sur chaque

page. L'aspect principal du site fut décidé par Roman, malheureusement il a quitté notre groupe. Donc ce fut Priscillien qui continua le site sur ces bases.

Les rubriques du site sont donc présentées ci-dessous :

1. Le projet Cette partie sera consacrée à la présentation du projet, ainsi il y est dévoilé tout le développement du jeu avec l'histoire de celui-ci, mais aussi toutes les armes et le principe de jeu.
2. Le groupe Dans cette section se trouve la présentation des membres comme vous vous en doutez. On y retrouve aussi leur adresse mail.
3. Téléchargement Dans cette rubrique seront exposés au téléchargement tous les fichiers que l'on aura cru bon de donner. Ainsi nous pourrons y retrouver le cahier des charges et les différentes soutenances dans tous les formats que nous avons, y compris une version latex. Dans cette section se trouve aussi une version light du jeu comme demandé. Celle-ci ne comporte qu'une map. La version complete est aussi disponible. Pour que le visiteur ne soit pas surpris nous indiquons la taille des différents fichiers à coté du lien.
4. Screenshot Ici sont proposés des captures d'écrans des maps, des voitures et des armes créées. Le visiteur peut donc se faire de lui même une idée sur l'avancé de notre projet.
5. Liens Cette rubrique est consacrée à tous les sites que nous trouvons utiles de recenser, ceux que nous avons utilisés pour delphi, opengl et autres. Mais cette rubrique comportera aussi des liens plus personnels sur les sites que nous apprécions.

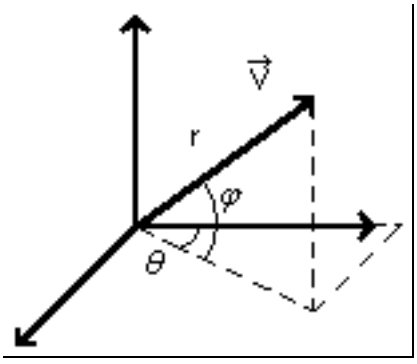
Chapitre 3

Premiere soutenance

3.1 Travail de joël

3.1.1 Création d'un objet vecteur polaire

Basée sur la logique physique, la création d'un objet vecteur est principale pour la gestion des déplacements et de l'orientation :



On utilise les vecteurs polaires pour plus de simplicité dans les transformations comme la rotation de la voiture qui vous est expliquée dans la section qui traite d'OpenGL. Mais, il est évident que lors des calculs pour les collisions et l'IA, il nous faudra convertir les coordonnées polaires du vecteur en cartésien.
Pour cela, nous utilisons les équations suivantes :

$$V_x = V_{norme} * sin(V_{teta})$$
$$V_y = V_{norme} * sin(V_{fi})$$
$$V_z = V_{norme} * cos(V_{teta})$$

Il n'y a pas d'erreur sur les sinus et les cosinus. En effet, nous considérons que lorsque téta vaut zero, les coordonnées du vecteur valent (0,0,1). cela est dû à l'orientation du repère d'OpenGL.

3.1.2 Création d'un objet position

Cet objet contient trois coordonnées qui représentent un point dans un repère à trois dimensions. Les procédures qui y sont liées sont l'initialisation, la translation par vecteur polaire et la translation par vecteur cartésien.

3.2 Travail de Priscillien

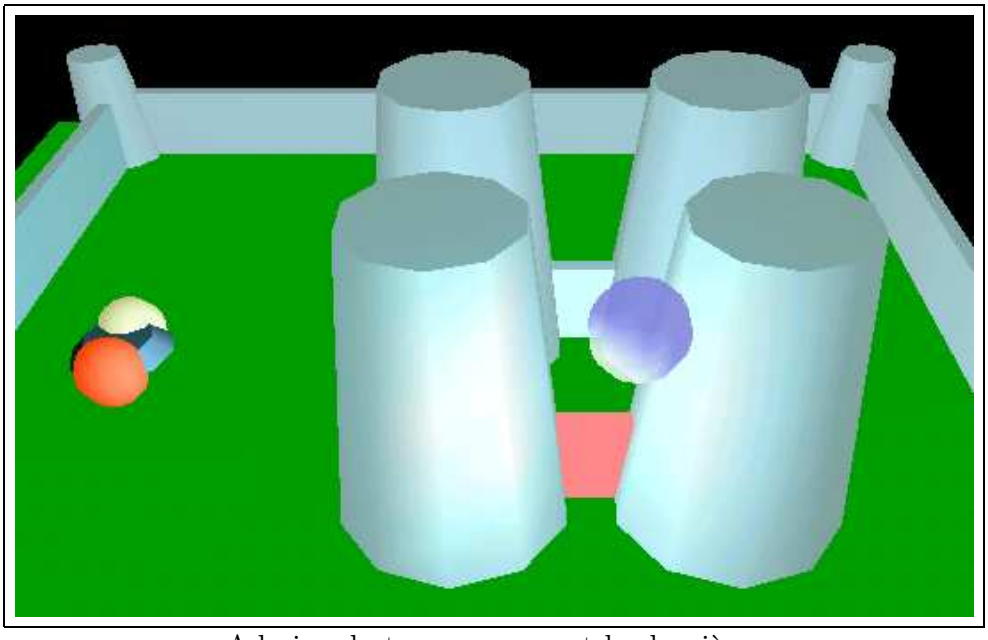
Comme pour toute chose il y a eu un commencement. Au début il a fallu effectuer des recherches concernant Delphi et Opengl.
Delphi tout d'abord car il s'agit d'un langage de programmation que je ne connaissais pas du tout. Mis à part quelques notions de basic, je ne voyais pas de quoi il en retournait. Donc il a fallu rapidement s'y mettre. Ensuite comme j'étais chargé de l'affichage avec Joël, nous nous sommes intéressés à OpenGl. Cette librairie graphique nous fut chaudement recommandé et beaucoup de personnes nous avançaient comme argument sa simplicité d'emploi. Nous avons quand même comparé les librairies Direct3D et OpenGl avant de nous lancer. Au premier abord Direct3D est plus fourni, mais malheureusement il est spécifique au système d'exploitation Microsoft. Cela nous a posé un petit problème car il est géré différement suivant que l'on soit sur Windows "standard" ou sous Windows NT. De plus cette API utilise un système de couche compliqué. Au fur et à mesure des versions une couche est rajoutée. Donc pour faire tourner notre application, il nous aurait fallu tenir à jour la version de DirectX. De nos jours, la plupart des ordinateurs sous Windows possèdent la version 7 ou supérieure. Lorsque l'on compare OpenGL et Direct3D sur ce point-là, on remarque que la plupart des constructeurs de cartes graphiques ont adapté les pilotes de leur matériel à ces librairies. Or certains, de plus en plus d'ailleurs, préfèrent optimiser leur carte pour OpenGL. Certains proposent même des fonctions spécifiques. La librairie graphique OpenGL, par contre n'évolue pas aussi souvent. Actuellement nous en sommes à la version 1.3. Cette librairie ayant à la base été dévelopée par des professionnels, notre choix s'est donc porté sur OpenGL. En effet Après avoir initialisé la fenêtre, nous avons rapidement pu afficher des primitives à l'écran.
Après cela, j'ai effectué des modifications de types dans le programme. En effet au départ nous avions juste un cube qui s'affichait et qui était capable

de se déplacer sur action de l'utilisateur. Il a fallu prendre en compte le fait que certains serait immobile comme la Map et d'autres mobiles, par exemple les voitures. De ce fait, fut créé l'objet Voiture. Tout objet affiché possède ses propres caractéristiques comme sa position et sa forme. La forme est obtenue par chargement du fichier ASE correspondant. Sinon il possède des méthodes comme avance, droite qui font respectivement avancer la voiture et tourner la voiture à droite. Le principe de la programmation objet est très utile. Nous aurions pu créer un type enregistrement pour les voitures et appeler des procédures à part, mais cela n'aurait eu aucun intérêt vu qu'il aurait fallu que ces procédures prennent en paramètre l'enregistrement en entier. Cela aurait été un peu lourd si l'on n'avait juste à modifier qu'un champ sur tous ceux composant l'objet.

3.3 Travail de Quentin

Apprentissage de l'environnement Delphi, de la syntaxe (objet...). Utilisation de différents types d'OpenGl pour Delphi tels que MnOgl et GlScene. Abandon de ces composants parcequ'il n'y avait aucune possibilité de modifier leurs comportements, et qu'ils ne créaient aucun source opengl.

Comme un beau dessin est toujours plus explicite qu'un gros discours, voila une capture d'écran.



Admirer la transparence et les lumières

Chapitre 4

Deuxième soutenance

4.1 Travail de joël

4.1.1 Création d'un loader de fichier ASE

Pourquoi les fichiers ASE

Afin de pouvoir afficher simplement des objets 3D, il semble nécessaire d'utiliser un format externe. 3D Studio Max est bien utile pour créer ces objets. Nous pouvons les exporter sous plusieurs formats. Nous avons d'abord pensé au format 3DS qui a une taille correcte mais la conversion des flottants est difficile car nous ne connaissons pas la manière dont ils sont codés en binaire. C'est pour cela que nous avons choisi les fichiers ASE.
(Au jour où nous tapons le rapport de soutenance finale, nous avons les connaissances mais pas le temps pour créer un loader de fichier 3DS).

Description des fichiers ASE

La structure des fichiers ASE est très pratique car elle autorise une lecture séquentielle (ligne par ligne). Pour apprécier sa structure, il vous suffit d'un logiciel de traitement de texte bien pratique du genre Notepad.

Ce fichier est certes très volumineux, mais sa structure est lisible dans un simple éditeur de texte comme WordPad, chose qui est impossible avec des fichiers au format 3ds.

Création des différents types

Afin de pouvoir utiliser les donnés stockées dans le fichier ASE, nous avons défini plusieurs types qui sont assez puissants car leur structure correspond à celle du fichier. Voici donc les différents types :

```
pVertex = ^TVertex;
TVertex = record
  x, y, z : glFloat;
  Next : pVertex;
  end;
```

Il s'agit des coordonnées d'un point dans l'espace. Bien sûr, ces coordonnées sont déterminées par rapport au repère d'origine de l'objet. 'Next' est le pointeur sur le vertex suivant.

```
pFace = ^TFace
TFace = record
  v : array[1..3] of pVertex;
  TextCoord : array[1..3] of pVertex;
  U  : TNormale;
  Next : pFace;
  end;
```

Ce type est utilisé pour déterminer les différentes faces de notre objet. Ces faces sont toutes des triangles et sont donc composées de trois vertexs. 'TextCoord' sont les coordonnés de texture qui sont très importantes car cela nous permet déterminer la taille des texture à appliquer sur la face : homothetie, torsion ...

```
TNormale = record
  x, y, z : glFloat;
  end;
```

Ce type correspond au vecteur normal d'une face. Il sera utilisé lors du traitement des lumières, des collisions et de l'IA.

```
pTexture = ^TTexture;
TTexture = record
  Id : GLuint;
  Next : pTexture;
  end;
```

Comme son nom l'indique ce type permet de connaître la texture appliquée au mesh. 'Id' est son identifiant. En effet, lors du chargement de la texture en mémoire, il faut garder un pointeur sur cette texture.

```
pMesh = ^TMesh;
TMesh = record
  VertexHead : TVertex;
  VertexQueue : pVertex;
  FaceHead : TFace;
  FaceQueue : pFace;
  CoordTextHead : TVertex;
  CoordTextQueue : pVertex;
  Texture : pTexture;
  Next : pMesh;
  end;
```

Une mesh est un sous-ensemble de l'objet principal. Par exemple un objet peut être composé d'un cube et d'un cylindre. Il possédera alors deux meshs qui sont le cube et le cylindre. Un mesh est en fait une forme géométrique simple. Il rassemble donc une texture, des faces et les vertexs s'y rattachant.

```
pObjet = ^TObjet;
TObjet = record
  MeshHead : TMesh;
  MeshQueue : pMesh;
  TextureHead : TTexture;
  TextureQueue : pTexture;
  end;
```

Comme évoqué précédemment, un objet est composé de meshs. Donc le type objet n'est qu'un ensemble de meshs et de textures utilisé par l'objet.

La plupart des types sont gérés dans des listes chaînées grâce à l'usage des pointeurs. Cela a un double avantage. Nous ne savons en effet pas au départ quel sera le nombre d'éléments de chaque type. L'usage que nous faisons de

la liste chaînée nous permet d'allouer de la mémoire pour chaque élément. De plus, la plupart des opérations que nous effectuons sur ces types se font dans un ordre précis que la chaîne permet de restituer sans problème : la liste chaînée est parcourue dans un seul sens (vers le bas).

Extraction des vertexs

Pour extraire les vertexs, on parcoure le fichier de ligne en ligne tant que l'on ne trouve pas la chaîne '*MESH_VERTEX_LIST{'. Une fois trouvée, on continue à le parcourir mais en chargeant les vertexs en mémoire jusqu'a se que l'on trouve '}' qui indique la fin du bloc.

La fonction convertfloat convertit les chaînes représentant des flottants en flottant. Le premier paramètre est une chaîne de caractères et le deuxième un entier représentant l'emplacement du flottant(si 1 alors premier flottant, si 2 alors ...).
L'axe Z et Y ont été inversés car les axes d'OpenGL et de 3D Studio Max ne sont pas les mêmes.

Extraction des normales

Lors de la création de nos algos de collision à la troisième soutenance, nous nous sommes rendu compte que les normales fournies dans les fichiers ASE sont erronées. Ce défaut a permis de résoudre un autre problème qui est la taille volumineuse des fichiers ASE. En effet, en supprimant les normales, on a pratiquement divisé par deux la taille des fichiers ASE. Le calcul des normales sera donc expliqué plus loin, dans la gestion des collisions.

Extraction des faces

Le principe d'extraction de face diffère légèrement. Comme le chargement des vertexs à déjà été effectué, la face comprendra trois pointeurs sur les vertexs correspondant.

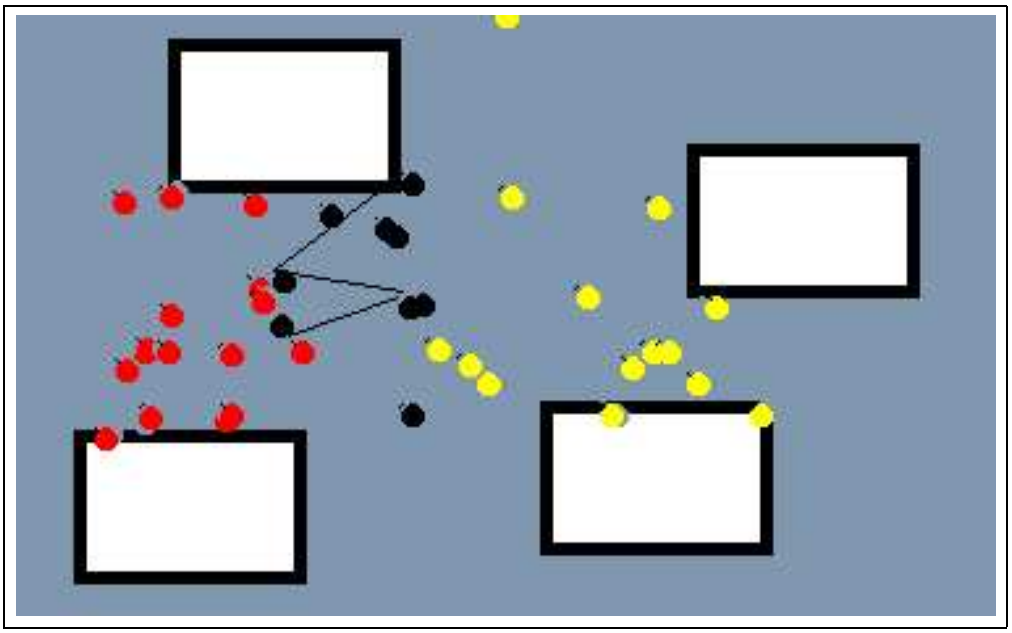
La fonction loadPvertex prend en paramètre l'identificateur du vertex, l'objet à créer, et renvoie un pointeur qui pointe sur le vertex correspondant.

4.2 Travail de Priscillien

Après l'initialisation effectuée à la première soutenance, je me suis occupé du changement de résolution et du mode plein écran. En effet, lors de la première présentation nous étions limités à la résolution de 800 par 600 et en mode fenêtré. Cela devenait gênant.
J'ai aussi commencé à m'occuper de l'apparence de notre lanceur. Ensuite je me suis occupé avec Joël du loader de fichier ASE.

4.3 Travail de Quentin

Création de la première dynamique des voitures. Les dessins étaient en 2D dessiné par Delphi. Les voitures étaient divisées en deux camps et se poursuivaient grâce à des champs de forces simples. Parce que la gestion des collisions n'étaient pas l'élément le plus primordiale, à ce moment là, les collisions ne se faisaient qu'en 2D avec de simples équations de droites.



Pas beau la 2D !

Chapitre 5

Troisième soutenance

5.1 Travail de Joël

5.1.1 Introduction

Il m'a semblé important de remettre la gestion des collisions car la version précédente était incomplète, et par conséquent, incompréhensible. On y a donc rajouté quelques images et corrigé les erreurs. Dans ce chapitre, Vous pourrez trouver des solutions aux collisions. Mais pour cela, il faut avoir certaines bases mathématiques : Eh oui ! L'informatique, c'est de la logique et des maths. SNIFF !
Trêve de plaisanterie, nous allons revoir rapidement les bases.

5.1.2 Bases mathématiques

Calcul de la distance entre deux points

Soit $A(x_a, y_a, z_a)$ et $B(x_b, y_b, z_b)$ deux points, la distance séparant ces points est égale à :

$$D = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2}$$

Calcul des coordonnées de la normale à une face

Pour déterminer la normale, il faut faire le produit vectoriel de deux vecteurs différents, non nuls et qui appartiennent au plan. Soit $A(x_a, y_a, z_a)$, $B(x_b, y_b, z_b)$ et $C(x_c, y_c, z_c)$ trois points qui appartiennent à un plan P. On en déduit les vecteurs appartenant au plan :

$$\vec{V}\begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix} = \begin{pmatrix} x_a - x_b \\ y_a - y_b \\ z_a - z_b \end{pmatrix}$$
$$\vec{W}\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} x_a - x_c \\ y_a - y_c \\ z_a - z_c \end{pmatrix}$$

On effectue ensuite le produit vectoriel entre ces deux vecteurs :

$$\vec{U}\begin{pmatrix} x_u \\ y_u \\ z_u \end{pmatrix} = \begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix} \times \begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} y_v z_w - y_w z_v \\ x_w z_v - x_v z_w \\ x_v y_w - x_w y_v \end{pmatrix}$$

il faut maintenant que la norme de la normale soit égale à 1. Pour cela, il suffit de diviser chaque coordonnées par :

$$\sqrt{x_u^2 + y_u^2 + z_u^2}$$

Equation d'un plan en 3D

L'équation d'un plan est de la forme :

$$ax + by + cz + d = 0$$

Pour trouver les variables a, b et c il suffit de connaître les coordonnées de la normale à la face. Soit $U(x_u, y_u, z_u)$ la normale à la face, on a :
$x_u x + y_u y + z_u z + d = 0$
Pour déterminer d, il faut prendre les coordonnées d'un point appartenant au plan : Soit $A(x_a, y_a, z_a)$ un point appartenant au plan, on a donc :
$d = -(x_u x_a + y_u y_a + z_u z_a)$
Pour conclure :
Soit $U(x_u, y_u, z_u)$ la normale au plan et $A(x_a, y_a, z_a)$ un point appartenant au plan. L'équation du plan est :

$$x_u x + y_u y + z_u z - (x_u x_a + y_u y_a + z_u z_a) = 0$$

5.1.3 Equation d'une droite en 3D

L'équation d'une droite est de la forme :

$$\begin{cases} ax + by + c = 0 \\ dy + ez + f = 0 \end{cases}$$

La troisième variable z se déduit de ces deux équations. Nous n'allons pas expliquer dans les détails les différentes manières de trouver l'équation d'une droite, mais nous allons juste évoquer les données minimales :

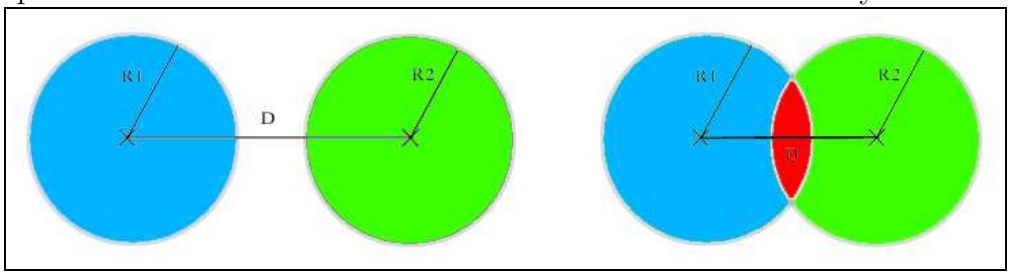
1. deux points,
2. un point et un vecteur.

5.1.4 Collision par bounding-sphère

A partir de cette section, nous rentrons dans la théorie, afin d'éviter que ce rapport ne se transforme en des équations mathématiques trop compliquées. Ceci dit, toute la théorie expliquée ici fonctionne et a été implémentée dans notre projet à cette étape de la soutenance.

collision bounding-sphère contre bounding-sphère

Cette collision est la plus simple à implémenter, mais elle est trés inexacte : les voitures ne sont pas des sphères et la différence risque de se voir. En pratique, il suffit de calculer la distance entre les deux centres des bounding-sphères et de tester si elle est inférieure à la somme de leurs rayons.



collision bounding-sphère contre face

Lorsque l'on fait des collision avec des faces, un pré-test s'impose. Il faut tester si les vertex d'une face sont toutes au-dessus de la voiture auquel cas il n'y a pas collision. On fait de même pour les faces trop en avant, en arrière,

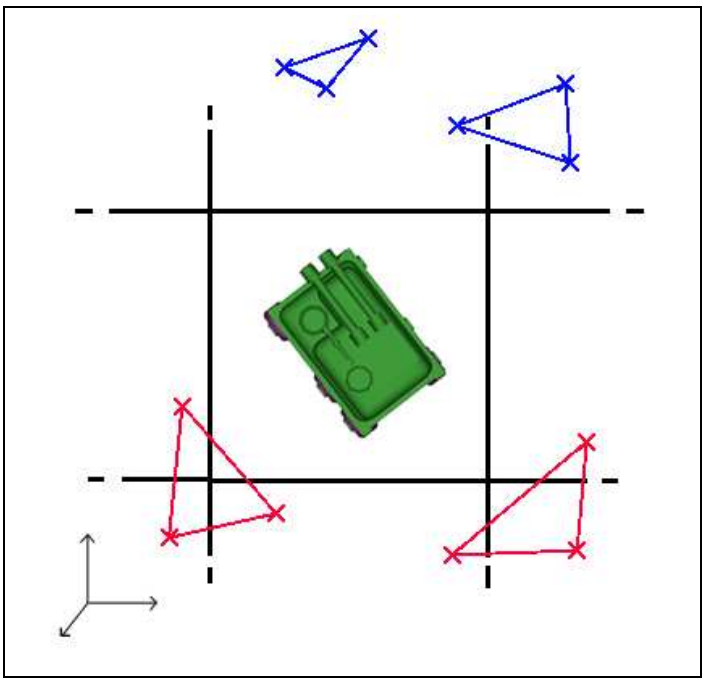
...
Avec les faces restantes, on fait les vrais tests. Il faut déterminer l'intersection I entre le plan que fait la face et la droite orthogonal à celle-ci passant par le centre de la sphère. Puis, il faut calculer la distance entre I et le centre. Si cette distance est inférieure au rayon alors le dernier test consiste à savoir si I se trouve à l'intérieur de la face. Ce test est expliqué à la fin de la section collision car elle est aussi utilisé pour les bounding-boxs.

5.1.5 Collision par bounding-box

Le principe est le même que pour la bounding-sphère. Il faut déterminer une boîte qui englobe l'objet. L'avantage est que la collision est plus fine, c'est-à-dire que les collision se font réellement entre les faces.

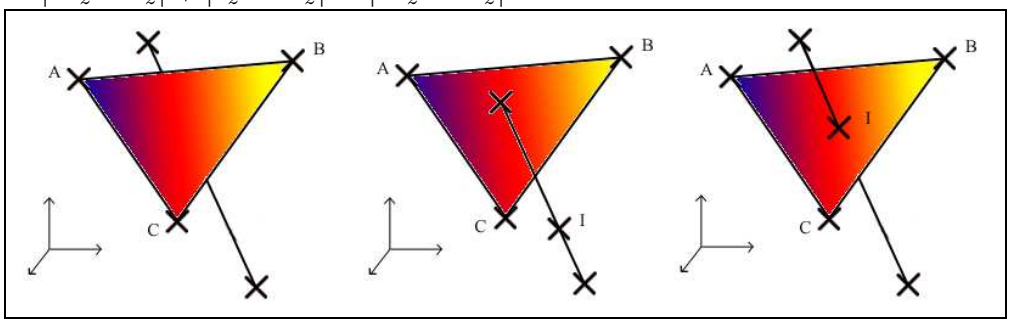
collision bounding-box contre face

Lorsque l'on fait des collision avec des faces, un pré-test s'impose. Il faut tester si les vertex d'une face sont toutes au-dessus de la voiture auquel cas il n'y a pas collision. On fait de même pour les faces trop en avant, en arrière, à droite et à gauche.



Avec les faces restantes, on fait les vrais tests. Il faut déterminer l'intersection I entre le plan que fait la face et la droite passant par deux points de

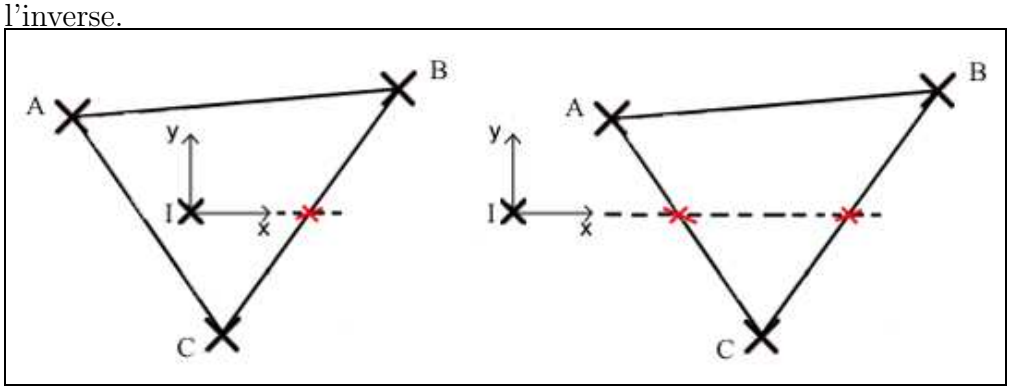
la bounding-box.
Ensuite, il est évident que le point I n'appartient pas obligatoirement au segment de droite de la bounding-box. Il est donc nécessaire de lever l'ambiguité : Si $|M_x - I_x| + |I_x - N_x| = |M_x - N_x|$ et $|M_y - I_y| + |I_y - N_y| = |M_y - N_y|$ et $|M_z - I_z| + |I_z - N_z| = |M_z - N_z|$ alors c'est à l'interieur.



Le dernier test consiste à savoir si I se trouve à l'intérieur de la face. Ce test est expliqué dans la prochaine partie.

5.1.6 Determiner si I appartient à une face

Il faut tout d'abord s'implifier le code en se ramenant sur un repère en deux dimensions. Pour cela, il faut projeter la face et le point I sur le plan (Ox,Oy), ou (Oy,Oz), ou (Oz,Ox). On détermine ce plan grâce à la normale à la face. Si le maximum des coordonées de la normale est sur l'axe Ox alors on projette sur (Oy,Oz), ...
Une fois projetté, il faut soustraire les coordonnés de I à celle de la face, puis determiner le nombre de coupure que font les segments formant la face avec la demi-droite $[Ox)$. Si ce nombre est paire alors, il y a collision sinon c'est l'inverse.



5.1.7 conclusion

Le code est long à tapé mais la rapidité et la qualité s'en voit améliorée.

5.2 Travail de Priscillien

5.2.1 L'héritage

A la fin de la deuxième soutenance nous avions un objet (une voiture) qui pouvait bouger dans son environnment. Encore fallait-il que le joueur puisse agir avec le monde. Donc plusieurs autres objets furent créer, à commencer par le joueur. Le joueur a plusieurs caractéristiques. Il possede dans ses champs une voiture, son niveau de vie, et d'autres caractéristiques utiles pour qu'il puisse intéragir avec les autres joueurs. Les différents éléments qui peuvent interagir sur le joueur sont les bonus et les armes. Bien sur il y a aussi les collisions avec la map et les autres joueurs. Pour les armes et les bonus nous avons du créer des nouveaux objets. Ces objets ont comme particularité qu'ils sont gérer avec l'héritage. Ils ont tous un ancêtre commun. Cela permet de gérer differents objets dans la même liste chainé. Pour exemple dans la liste chainée des armes, le premier élément peut être un missile et le second une mine. L'héritage permet aussi d'éviter les redondances de codes. C'est dans notre objet arme que nous mettons les différentes procedures de test pour les collisions et non dans le missile et dans la mine. On a appliqué cette méthode afin de mieux structurer notre code. la notion d'héritage permet aussi de déclarer des procedures dans l'ancêtre commun mais qui auront un auront des actions différentes suivant le fils utilisé. Cela a été très utile lorsque l'on a déclaré les actions des bonus. L'exemple le plus concret est la différence entre un bonus qui donnes des munitions et celui qui donne des points de vie. Le bonus qui donnes des munitions agira sur le champ arme du joueur et celui qui donnes des points de vie agira sur le champ vie. La procédure appelée ici est la procédure action du bonus. Bien sur comme expliqué précedemment nous l'appelons à partir de l'ancêtre. Cela évite de faire des tests à répétition pour savoir à quel bonus nous avons à faire. De plus a fin d'éviter de passer trop de paramètre dans la procédure nous avons préféré lui mettre en paramêtre un pointeur sur le joueur qui collisionne avec le bonus. De ce fait la procédure à directement accès au champ du joueur qu'elle doit modifier et on évite de mettre des paramètres inutiles comme le niveau d'essence pour la procédure qui affecte les points de vie.

5.2.2 Les bonus

Les principaux évènements à gérer vis-à-vis des bonus sont leurs apparitions et leurs prises d'effet. Pour leurs apparitions, nous avons décidé de mettre les coordonnées de ces points dans un fichier à part de la carte pour des raisons de commodité. Nous considérons en effet qu'il est inutile de surcharger les fichiers des maps. Nous aurions pu mettre des objets d'une forme prédéfinie dans le monde, invisibles au joueur et que l'on peut traverser, mais cela aurait fait effectuer des calculs supplémentaires alors que nous possédons déjà les coordonnées de ces points. De plus, il aurait fallu considérer des objets avec lesquels il y a des collisions et des objets avec lequels il n'y en a pas. Cela aurait alourdi le code pour rien.
Pour les effets des bonus ils se font toujours lorsqu'il y'a une collision entre un bonus et un joueur (sa voiture). Nous considérons dans notre jeu que les effets d'un bonus sont prioritaires par rapport à ceux des armes. Cela permet a un joueur d'éviter in extremis la mort lorsque il arrive a atteindre un bonus de vie et qu'un missile lui fonce dessus. Nous savons bien sur que dans la rlité cela serait impossible. Mais avez vous déjà vu une voiture tirer un missile ? personelement si c'était le cas je m'inquièterais beaucoup.
Les différents bonus que nous avons mis dans le jeu sont des bonus de vie, d'essence, et de munitions. Nous n'avons pas mis de bonus de type exotiques, cela n'aurait pas collé avec l'ambiance que nous avons voulu faire ressentir au joueur. Les bonus sont gérés dans une liste chainée crée à partir de l'ancêtre commun à tous les bonus. Cette liste chainée est placé dans la boucle de jeu principale.

5.2.3 Les armes

Pour les armes, nous avans procedé de la même manière que les bonus nous avons définis un type général. Ensuite, nous avons définis les méthodes particulières de chaque arme dans le type correspondant. Ils sont eux aussi gérés dans une liste chainée. Sauf que celle-ci au lieu d'être rattaché à la boucle principale est rattaché au joueur. On évite de cette façon de mettre un champ dans le type arme qui indique qui a lancé l'arme en question. De la même manière cela nous a éviter de faire un test lorsque un joueur lance une arme. Nous considérons dans notre jeu que les seuls joueur suicidaires qui peuvent exister sont ceux qui s'éperduent à foncer contre les murs. En clair nous considérons qu'un joueur n'est pas afecté par les effets de ses propres armes.
Nous avons défini six armes. Celles-ci sont des balles, des missiles,des mines et

un laser. Le laser fut l'idée de Quentin mais fut réadapté afin de correspondre au type commun. Nous avons distingué trois types de misssiles. Chacun a sa particularité. Nous avons le missile classique qui inflige des dégâts, un autre qui enlève de l'essence au joueur touché, et le dernier, et non des moindres, un missile dirigé par le joueur. C'est celui-ci qui a posé le plus de problèmes étant donné qu'il faut changer le point de vue du joueur. Cela est expliqué dans le chapitre suivant.
Pour des raisons de réalisme, nous considérons que lorsqu'un joueur passe l'arme à gauche, les armes qu'il a lancées subsistent. Mis à part le missile que l'on peut diriger, cela est tout-à-fait normal. Une mine posée n'a aucune raison de disparaître sauf lors d'une explosion.

5.2.4 La caméra

La caméra, autrement dit le point de vue du joueur est un élément essentiel dans un jeu. Dans le notre nous avons décidé que lorsque le joueur est dans une voiture il dispose de trois point vues possibles. Ces vues sont la vue intérieure, une vue extérieure rapproché et une vue extérieure éloignée. Nous avons danc défini un objet caméra adapté à cela. Comme évoqué précedemment nous avons un missile que nous pouvons diriger. Lorsque celui-ci est lancé la voiture stoppe et le point de vue de la caméra se fixe sur le missile. Le joueur ne pourra donc s'en prendre qu'a lui même s'il n'atteint pas sa cible.
Afin d'actualiser la position de la caméra nous effectuons unn parcours de la liste des joueurs jusqu'a trouvé le joueur actif. A ce moment les coordonné de la caméra sont calculés suivant le point de vue.

5.2.5 L'affichage des informations destinées au joueur

Au cours du jeu, le joueur, en plus de voir les autres joueurs, doit être au courant de son statut. Ici le statut du joueur est défini par son armement, son niveau d'essence et de vie. Il doit aussi connaître d'autres informations tel que le nombre de morts et le nombre de joueurs restants dans chaque équipe. Pour y arriver nous affichons du texte à l'écran a coté du symbole correspondant. Afin d'afficher une police de caractère à l'écran nous avons décider d'utiliser des fonctions de windows qui permettent de mettre chaque caractère dans une liste d'affichage OpenGL. Contrairement aux autres objets de la scène ces caractères ne sont pas constitués de polygones. Au début nous pensions créer des textures pour chaque caractère mais cela aura sur-

chager la mémoire de textures d'une part et aurait pris plus de temp de calcul étant donné que l'on aurait du y appliquer de l'Alpha Blending.
Pour afficher les caractères nous avons définis une grille totalement indépendante de la résolution. Néanmoins à chaque résolution correspond une taille de la police différente. Nous avons réajusté les postions de tel façon que les indiquatrions apparaissent toujours au même endroit.
Pour les armes, seulle la quantité restante de l'arme sélectionné est indiqué par un nombre et cela toujours au même endroit. Pour toutes les armes par contre nous afichons une jauge verticale qui indique la quantité restante. Lorsque le niveau est au maximum la jauge prend toute la largueur de l'espace qui lui est réservé. Nous avons fait en sorte que cela apparaisse au premier coup d'oeil au joueur. Pour distinguer l'arme selectionnée des autres nous la mettons en surbrillance. Nous indiquons aussi le numéro de l'arme ce qui permet au joueur d'y accéder directement en appuyant sur le numéro correspondant.

5.3 Travail de Quentin

5.3.1 Gestion du son

Le jeu SpeedWay se voit maintenant doter de sons et de bruitages qui vont rendre le jeu plus agréable.

TurboSound

TurboSound est un composant de Delphi pour le son. Il contient des composants pour travailler sur les fichiers wave standard de Windows et les objets de Direct Sound. Il contient trois composants : TWave, TWaveMixer et TSoundBuffer. Les deux derniers sont les plus importants.

- *TWave* sert à encapsuler un fichier wave Windows. Mais les deux autres composants sont les plus importants.
- *TWaveMixer* charge DirectSound et indique si le chargement s'est bien effectué. Il permet de mixer les sons et d'en jouer plusieurs à la fois.
- *TSoundBuffer* est un objet Delphi qui permet d'accéder aux sons stockés sous forme TSoundBuffer dans DirectX.
  Grâce aux fonctions play, loop, volume, fréquence, balance, il permet de gérer les sons.

Malheureusement, TurboSound étant un shareware, demande, pour pouvoir être utilisé, de payer des droits aux auteurs.

Direct Sound

Lors de la création de la fenêtre, on crée un tableau de sons wave. Le programme va chercher dans un dossier les fichiers sons wave (pour simplifier le code source tous les fichiers ont le nom Sound+n,où n est un entier). Aprés avoir mis dans le buffer (en mémoire) tous les sons, on peut en lancer plusieurs simultanément en appelant le ènième son du tableau. On peut également arrêter le son comme modifier son volume, sa fréquence etc...

TMediaPlayer

DirectSound nécessitait que tous les sons soient en mémoire. Par contre MediaPlayer permet d'utiliser les sons à partir d'un CD-ROM sans avoir à les stocker en mémoire. Pour accéder au son, il suffit d'initialiser dans l'inspecteur d'objet en y affectant la valeur dtCDAudio à DeviceType, puis de créer le bouton 'jouer'. Un menu permet de chercher dans l'arborescence tous les fichiers wave. Ceci est possible grâce aux contrôles : *DriveComboBox*, *DirectoryListBox* et *FileListeBox* DriveComboBox permet de spécifier un disque et envoie le chemin à DirectoryListBox, qui lui affiche tous les dossiers possibles du lecteur sélectionné. Il envoie le chemin à FileListeBox qui grâce à un filtre affiche tous les fichiers wave.

5.3.2 Initiation à la 3D

La création de voitures et d'immeubles sous 3D Studio Max et l'apprentissage d'OpenGl m'a permi de passer de franchir le pas de la 2D vers la 3D.

New York dans la nuit !

Chapitre 6

Soutenance finale

6.1 Travail de Joël

6.1.1 Déplacement de la caméra

Les déplacements de la caméra ont été faits lors de la première soutenance mais un problème s'est posé lorsque l'on a créé la carte. La caméra traversait les murs, nous avons donc modifié sa position grâce aux collisions implémentées à la soutenance précédente :
On calcule d'abord la position normale de la caméra, puis on teste les différentes collisions entre le segment caméra-voiture et les faces de la carte. Il reste à trouver la collision la plus proche de la voiture ce qui indiquera la position de la caméra.
ce principe a été implémenté et l'on peut y remarquer quelques effets indésirables tels que :

1. Le saut de la caméra lors de la rotation de la voiture dans un coin aigü de la carte.
2. La possibilité de voir la pièce d'à côté lorsque la caméra est en collision et que la voiture est pratiquement orientée parallèlement au mur.

Le deuxième problème peut se résoudre facilement, la solution est de créer une bounding-box ou une bounding-sphere autour de la caméra.

6.1.2 Quelques procédures pour l'IA

Lorsque nous avons développé les collisions, nous nous sommes rendus compte que l'on pouvait les utiliser pour l'IA.
Pour savoir si une voiture est vue par une autre, il faut tester s'il y a une face

entre les deux voitures. Le principe est toujours le même pour les bonus, ...

6.1.3 L'emplacement des joueurs et des bonus

L'emplacement des joueurs et des bonus a été déterminé grâce à 3D Studio Max et un petit programme que nous avons créé pour nos besoins. Il prend deux coordonnées x et z et passe au suivant. Il y a environ 20 coordonnées maximum car le nombre de joueurs ne dépassent pas 19. De même pour les bonus qui sont au nombre de 10. On a donc un tableau de taille 20.
Quand une voiture prend un bonus, il disparaît de sa position et réapparaît à une position libre pour le moment. La possibilité que l'on ait deux bonus au même endroit est donc à exclure. Cette méthode n'est pas très jolie mais cela rend le programme plus simple. En effet dans les Doom-Like, on utilise une temporisation qui est plutôt embêtante à implémenter.

6.1.4 Responsabilité du chef de projet

Le travail que j'ai fait pour cette soutenance finale n'est pas très visible car il m'a fallu faire du travail 'de fond'. Je m'explique, j'ai recodé les choses illogiques sur tout le programme, refais la présentation, arrangé le code pour qu'il soit le plus clair possible... Je considère aussi toutes les destructions des listes chaînées comme un travail invisible mais nécessaire. Pour moi, ça a été le plus gros du travail qui m'a paru le plus dur. La présentation au démarrage, les sons à placer au bon endroit, la finalisation, sont souvent les plus longues choses à faire !

6.2 Travail de Priscillien

6.2.1 Finition

Les informations destinées au joueur

Avant de rendre le projet, il nous a fallu faire des optimisations dans le code. Nous tenons à ce que notre jeu soit beau graphiquement et qu'il puisse fonctionner sur la plupart des ordinateurs actuels. La première optimisation q'il nous a fallu réaliser fut pour l'affichage des informations pour le joueur. En plus d'afficher, le nombre d'arme et tout cela, nous avions décidé d'appliquer un fond pour que le joueur se repère facilement. Au départ il faisait tout l'écran, et nous avons constaté des ralentissements majeurs sur les petites machines. Afin d'économiser du temps de calcul nous avons décidé de

découper l'image en des parties plus petites et de n'effectuer de l'Alpha Blending que sur ces petites parties. L'effet s'en est directement ressenti car les parties utiles de l'image occupaient nettement moins de surface que l'image toute entière.

Le loader ASE

Contrairement à la police de caractère, nous n'utilisions pas les listes d'affichages lorsque nous chargions les objets à incorporer dans notre scène 3D. Donc la procédure de chargement, ainsi que le type des objets furent modifiés de façon à y intégrer la liste d'affichage. Les listes d'affichage ont plusieurs avantages. Les temps de calculs sont beaucoup moins longs. Ces listes sont compilées, puis le résultat est stocké en mémoire. Elles sont dans un format directement compréhensible par la carte graphique. Dans le programme, l'affichage des objets est aussi simplifié. Il suffit juste de faire comme les textures et d'appeler l'index correspondant.

6.2.2 Effet

- L'alpha blending
  L'alpha blending est un procédé qui permet de réaliser des effets de transparence. Nous l'avons configuré de telle façon que lorsque nous l'activons, la couleur noire est supprimée. Nous utlisons principalement l'alpha blending lorsque nous affichons les informations aux joueurs et lors des explosions. Les explosions sont gérées avec un moteur de particules.

- Le fog
  Le fog est un effet de brouillard. Nous l'utilisons afin que le joueur n'aperçoivent pas un fond noir lorsque des objets sont en dehors de son champ de vision. De cette façon nous faisons croire que les objets apparaissent progressivement. Pour que l'effet soit réaliste nous avons utilisé un fog exponentiel et nous nous sommes arrangés pour qu'il ne soit visible que dans l'horizon.

- Le moteur de particules
  Nous utilisons le moteur de particules pour les effets d'explosions. Lorsqu'une arme explose, nous ne créons pas moins de 500 particules que nous envoyons dans des directions aléatoires. Le principe d'un moteur

de particules est très simple. On agit en trois phase une fois que l'initialisation est réalisée. Première phase on supprime les particules inactives. Deuxième phase, on actualise leurs coordonnées. Troisième phase, on affiche les particules. Nos particules sont très simples à afficher il s'agit de deux plans formant une croix. Afin de donner l'impression d'un objet, nous utilisons l'alpha blending. Une quatrième phase peut s'intercaler après l'actualisation des coordonnées, il s'agit de la création de nouvelles particules. Nous ne l'avons pas fait car sur certaines machines on peut avoir l'impression qu'il y a eu plusieurs explosions alors qu'une seule est effectuée. Cette phase est en général utilisée lorsque l'on a envie d'un effet de longue durée.
Afin de vraiment peauffiner l'effet, la particule vire du rouge à l'orange. Cela fut un peu dur à réaliser étant donné que l'on changeait la couleur de la texture. Cela n'a rien à voir avec l'alpha blending qui réalise un fondu. Autre problème qui s'est posé c'est lorsque nous avons affiché les particules. Certaines n'apparaissaient pas sur fond blanc. Après avoir recherché il nous est apparu que c'était un problème avec le test de profondeur. Nous avons donc désactivé le test de profondeur lorsque nous affichons les particules. A ce moment les particules étaient affichées. Trop bien même, elles étaient même visibles à travers les murs. Donc après quelques recherches nous avons trouvé une fonction qui nous résolvait le problème. Cette fonction ne permettait qu'un accès en lecture seule au tampon de profondeur. Donc, lorsque la particule est dessinée, il n'y a pas besoin de désactiver le test de profondeur, mais juste de le mettre en lecture seule. Cela permet en plus d'afficher la particule correctement, de ne pas l'afficher quand elle se situe derrière un mur visible par le joueur.

6.3 Travail de Quentin

6.3.1 Scénario

Speed Way est un jeu doom-like en équipe, c'est pourquoi, au début du jeu on doit créer pour chaque équipe, un leader et un "squad" de soldats.

Maintenant que les "squads" sont préts et que les deux leaders ont une frénésie guerrière, on peut commencer le jeu : Les deux leaders vont se chercher, attirant avec eux tout leur régiment.

Lors de la bataille, si un leader est tué, alors la voiture qui a le plus tué (la première voiture de la liste) devient le leader (ça va de soit) et dirige

alors les survivants. Si une voiture (sauf les deux leaders) a trop perdu de vie, tente de se maintenir en vie en fuyant le champ de bataille.

Remarque : une amélioration possible est de simuler des combats Romains modernes, c'est à dire dans chaque régiment, un centurion dirige dix décurions, dont chacun d'eux s'occuppent également de dix hommes.

Le tir est simple à mettre en place, en effet si une voiture voit un ennemi qui n'est pas caché par un mur alors on lance un missille (un missille adéquat par rapport à la distance des voitures). Un problème est survenu : la fréqunece des tirs. En effet lorsqu'une voiture contôlée par l'ordinateur voit une autre, elle tire un trés grand nombre de missilles en un temps trop court. Pour être plus explicite : tout le stocke de munitions est écoulé en un seul coup (c'est ce qui s'appelle faire d'une pierre deux coups !). La solution est de faire tirer tous les n coups (les $n-1$ coups sont des coups fictifs).

6.3.2 Un peu de maths

Les voitures ne sont pas aveugles

Etant donné deux voitures en position A et B, on note (x, y) les coordonnées du vecteur $\overrightarrow{AB}$ et (v_x, v_y) les coordonnées du vecteur vitesse $\overrightarrow{V}$ de A l'angle $\theta = (\overrightarrow{V}, \overrightarrow{AB})$ est défini par

$$\cos\theta = \frac{xv_x + yv_y}{\sqrt{(v_x^2 + v_y^2)(x^2 + y^2)}} .$$

Pour qu'une voiture tire, il suffit, par exemple, que le $\cos\theta$ soit supérieur à 0.8.

La liaison des armées.

La dynamique des voitures est définie à partir de deux champs de forces exercées par B sur A :

- le premier est attractif et d'intensité lineaire avec la distance entre les xvoitures :
$$\overrightarrow{F}_a = k_a \overrightarrow{AB}$$
- le second champ est répulsif et d'intensité inversement proportionnel à la distance (r) entre les voitures :
$$\overrightarrow{F}_r = -k_r \frac{\overrightarrow{AB}}{r^2} .$$

On supposera les masses des voitures identiques égales à 1. Alors l'accélération (γ) de A due au champ de force exercée par B vaut

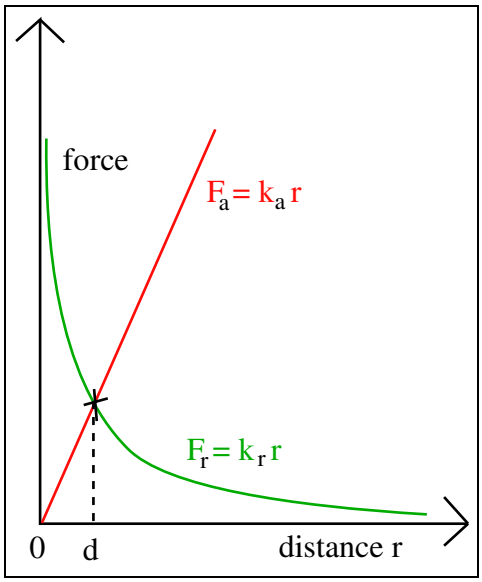
$$\overrightarrow{\gamma} = \overrightarrow{F}_a + \overrightarrow{F}_r \ .$$

On impose de plus au vecteur vitesse les contraintes $|v_x| \leq v_x^{\max}$ et $|v_y| \leq v_y^{\max}$ on en déduit la modification du vecteur vitesse v' lorsque A subit l'accélération due à B pendant une unité de temps :

$$v'_x := \min\left(\max\left(v_x + \gamma_x, -v_x^{\max}\right), v_x^{\max}\right),$$

$$v'_y := \min\left(\max\left(v_x + \gamma_y, -v_y^{\max}\right), v_y^{\max}\right).$$

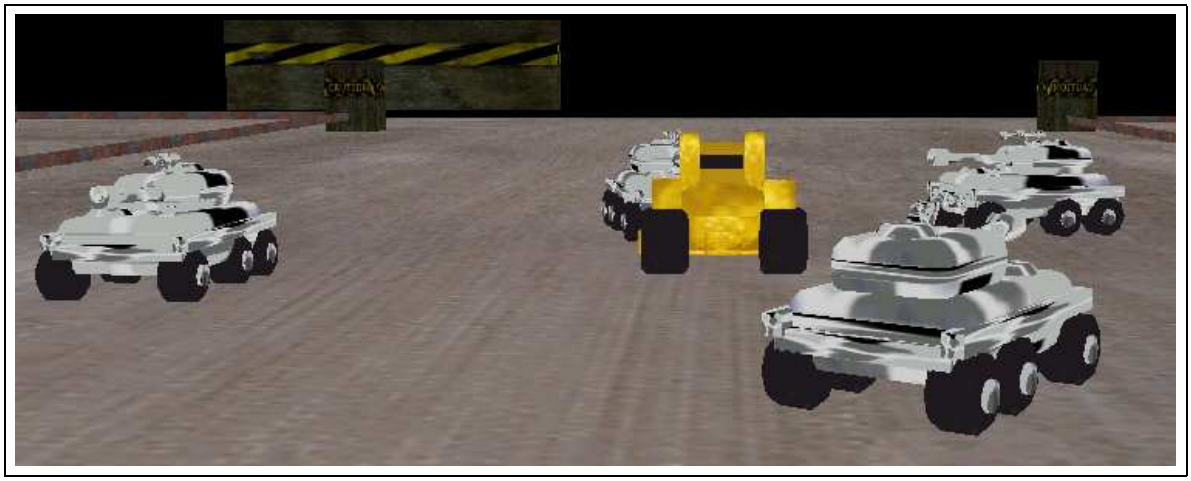
Pour que les voitures suiveuses restent à une distance d'environ d du leader on peut ajuster les coefficients k_a et k_r de façon à ce que la somme des deux champs s'annule à cette distance de B (voir figure **??**).



Calcul du lien entre k_a et k_r.

6.3.3 Dans le jeu ...

Les leaders attirent leur régiment avec un champ de force attractif et linéaire par rapport à la distance. Mais toutes les voitures se repoussent avec un champ de force répulsif inversement proportionel à la distance. Les deux leaders s'attirent avec un champ attractif différent du premier mais toujours proportionel à la distance.



Sir, yes sir !

6.3.4 Problèmes rencontrés

Pourquoi avoir utiliser des champs de force plutôt qu'autre chose ? Simplement pour éviter des problèmes du type suivant :

L'ordinateur doit poursuivre la première voiture ennemie dans sa zone de vision (note : on appelle zone de vision l'angle θ vu dans la section "les voitures ne sont pas aveugles"). Mais si l'ennemie sort de cette zone, alors il ne la suit plus, malgré qu'il soit à proximité. Pour éviter cet inconvéniant, il faudrait d'une part enregistrer la dernière position de la voiture, puis s'y diriger. Pour simuler ce phénomène, l'utilisation d'un champ de force est le bien venu.

6.3.5 Les effets speciaux

Blending

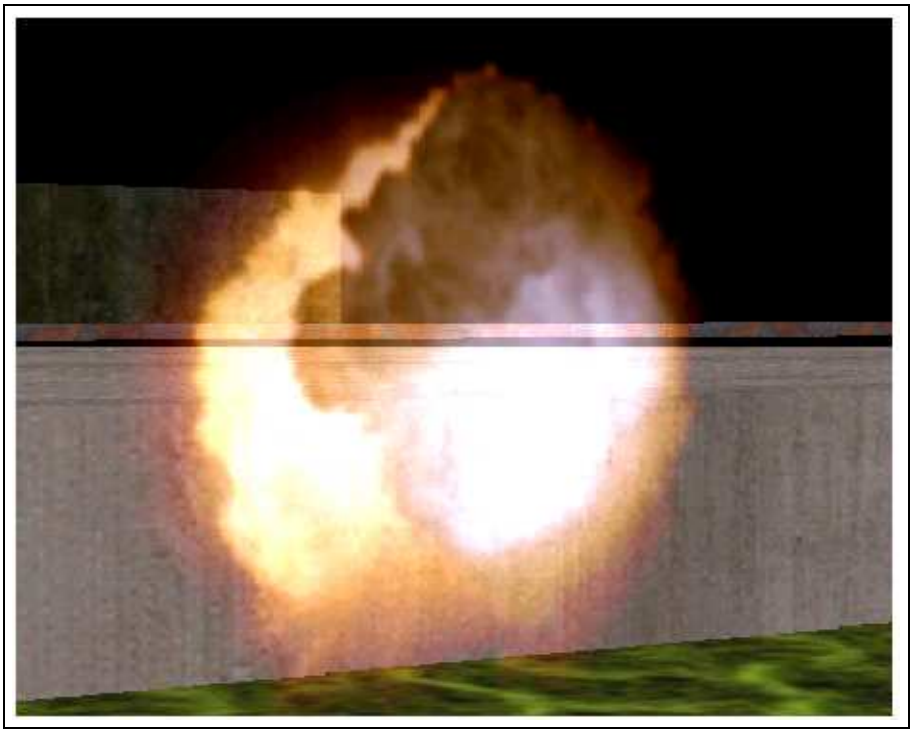
Le Blending est utilisé pour combiner les couleurs des nouveaux pixels que l'on vient de créer avec ceux déjà présents dans la scène. La combinaison des couleurs est basée sur la valeur de la quatrième couleur qui rend les pixels plus ou moins transparents (les trois premières couleurs sont).

Explosion sans particules

C'est le même principe que les dessins animés, on stocke toutes les images d'une explosion au cours du temps, puis on les affiche les unes apres les autre suffisament vite pour créer l'impression d'une vraie explosion. On utilise l'alpha blending pour rendre transparent tout ce qui est noir.

L'avantage majeur est la rapidité : d'une part on perd très peu de temps à creer des listes chainées de particules, qui ralentissent le temps de calcul de l'ordinateur et d'autre part ce n'est qu'une texture de plus à afficher ! Mais, le soucis majeur est que les images sont affichées sur un plan 2D que l'on doit corectement orienter par rapport au joueur. Pour résoudre ce problème on utilise deux plans croisés avec la même image d'explosion : ce principe est utilisé pour créer de arbres sans utiliser de polygones. Le deuxième soucis est que l'on doit mettre en mémoire énormement de textures sur le disque.

Les explosions sont des objets qui contiennent leurs positions dans l'espace, le type d'explosion, le numero de l'image à afficher et enfin la procedure qui permet d'afficher l'effet d'explosion. Au début, je ne pouvais que créer qu'une seulle explosion à la fois (due à une variable globale qu'il fallait incrémemter à chaque tour de boucle), mais depuis qu'elles ont été implementées sous forme d'objet, on peut en créer plusieurs simultannement. Ensuite, j'ai cré des listes chainées d'explosions pour faciliter .

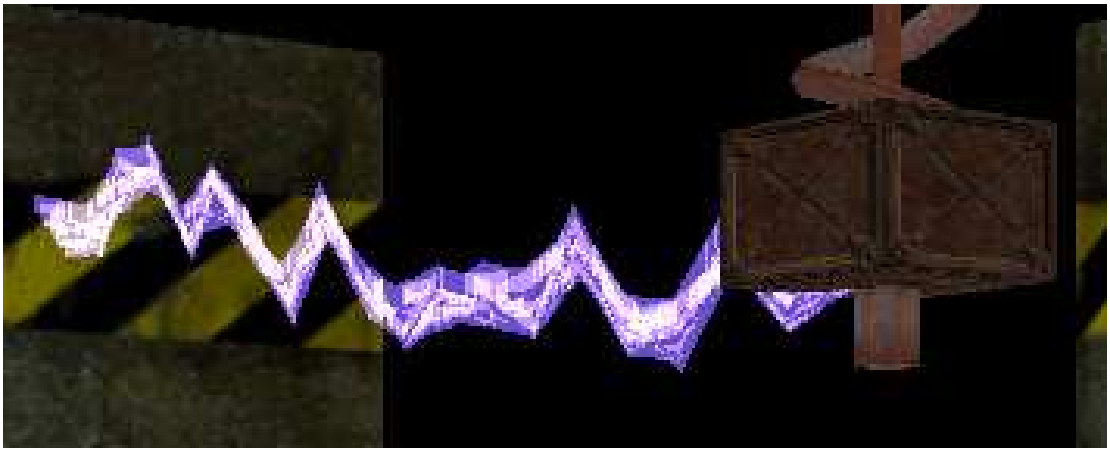


Ca doit faire mal !

L'Electricité facile sans électrocution

Même pour les effets d'electricités, l'utilisation du blending est impératif, bien qu'ici on n'affiche aucune texture. En fait, créer des triangles bleus en

les faisant zigzager de haut en bas. Les triangles sont semi-transparents, mais plus en affiche, plus ils deviennent blancs : ce qui nous donne un bel effet délectricité tout a fait innofensif.



C'est inoffensif

Eblouissement

Pour créer un effet d'éblouissement, il suffit de créer un rectangle blanc de longueur et de largeur de la taille de l'écran, puis de diminuer progressivement les trois couleurs de bases jusqu'à zéro.

6.3.6 Création de terrain

Cette partie n'a pas pu être implémentée. Il existe plusieurs méthodes pour créer un terrain aléatoire. Les plus courantes sont basées sur l'utilisation des fractales, bruits...

La première méthode que j'ai appris est d'utiliser une image quelconque en noir et blanc. La deuxième se base sur des polynômes 3D. Explication de la première méthode. Dans un premier temps, on génére un maillage carré dans le plan XY, puis chaque carré est divisé en deux triangles. Une fois le maillage créé, il n'y a plus qu'à affecter aux sommets de chaque patch une coordonnée en Z. En fonction de la position du sommet on établit une correspondance avec un pixel de l'image en niveaux de gris. Si le pixel correspondant est noir l'élévation du sommet sera minimale, s'il est blanc il sera maximale (principe des heightfields).

La deuxième méthode est d'utiliser des fonctions polynomiales. C'est le même principe que pour les "plot 3D" des logiciels mathématiques, la principale utilité est de pouvoir utiliser la courbe pour positioner correctement

les voitures en hauteur et dans la bonne orientation (selon la tangente à la courbe). Malheureusement, j'ai pu que tracer des fonctions telles qu'un cercle, mais je n'ai pas réussi à faire passer un polynome par plusieurs points.

Chapitre 7

Conclusion

7.1 Le plus important

Ce projet que nous avons effectué à trois, nous a impressionné. Le mot est un peu fort mais je peux vous dire que lorsque nous sommes rentrés à l'EPITA, nous n'aurions jamais pensé faire un projet aussi complet. En tant que chef de projet, j'ai été étonné par l'intérêt que nous y portions. En effet, aucun de nous n'a percu le projet comme une corvée.
Toutes les vacances précédant les soutenances nous ont servi à travailler sur le projet. J'entends par là que nous ne sommes pas partis en vacances ; nous sommes restés à Paris pour travailler. Ceci prouve que le travail n'a pas été une corvée.
Le départ de Roman était prévisible, c'est pourquoi nous ne lui avons pas donné des choses énormes à faire. Nous n'avons pas fait la gestion du joystick car cette idée venait de lui. En effet aucun des autres membres du groupe ne voyait l'utilité de ce genre de manette de jeu.

7.2 Le moins important

"Ofortunatos nimium sua si bona norente agricolas" Traduction : "Trop heureux, les hommes des champs s'ils connaissaient leur bonheur" mais je ne sais plus comment ç a s'écrit. Tirade tirée de la BD *Astérix chez les bretons*

7.3 Les remerciements ou comment noircir des feuilles

- Je remercie mon fidel ordinateur, qui grâce à lui, j'ai pu faire ce projet.
- A opengl pour faire de très beaux effets spéciaux.
- A delphi, qui a permis de créer la boucle du jeu avec toute la dynamique des voitures.
- Je ne remercie pas directX/DirectSound, car c'est un langage qui posséde une syntaxe horrible.
- Je ne remercie pas opengl parceque sa syntaxe pourrait être nettement mieux améliorée.
- Je remercie dans l'ordre le site d'Eraquilla, pour m'avoir appris comment créer ma première fenetre opengl, puis mes premières primitives.
- Je remercie un autre site dont j'ai oublié le nom de l'auteur (mais qui se reconnaitra peut-être pas) pour m'avoir appris à créer des effets simples pour Opengl.
- Je remercie le site Nehe, pour ses nombreux tutoriaux, mais je ne le remercie pas parce que je n'ai pu compiler aucun de leurs exemples.
- Je ne remercie pas le livre de réference d'Opengl car il est pour moi, tres moyen puisqu'il explique tres mal comment utiliser Opengl, mais je le remercie pour m'avoir insiter à apprendre opengl uniquement grâce au web et non sur les livres (mauvais rapport qualité/prix).
- Je remercie 3DSutdioMax, pour m'avoir permis de dessiner quelque belles voitures dans un excellent rapport qualité graphique/heure.
- Je remercie particulierement les erreures de virgule flottante pour un redémarage sistématique de la machine (pas un seul écran bleu). Il faut savoir apprécier les bons moments d'informatiques :
  - taper votre code, compiler,
  - plantage immediat sans message d'erreur,
  - redémarrer l'ordinateur (sans avoir compris où se situait le bug),
  - passer scandisk, (4 minutes par redemarage), lancer delphi,
  - trouver la soi-disante erreur, taper votre code, recompiler,
  - recommencer les étapes 3 à 5, pendant cinq fois, en pensant TRES fort que vous avez probablement trouver une erreur, que ce que vous venez de taper est sans faute et qu'il ne reste plus qu'une faute à trouver (il y a de l'espoir)...