École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
3e année
2011 - 2012

Rapport de projet d'Algorithme et de Langage C

Simulation du trafic urbain

Encadrants
Néron EMMANUEL
emmanuel.neron@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Lei SHANG
lei.shang@etu.univ-tours.fr
Yanxiao HU
yanxiao.hu@etu.univ-tours.fr

DI3 2011 - 2012

Version du 17 juin 2012

Table des matières

Table des figures

Introduction

Ce projet est développé dans le cadre de la formation de l'algorithme et le langage C en troisième année du département informatique. Le but du projet est de pratiquer les connaissances que nous avons acquises durant les séances de cours.

Pour cela, nous sommes demandés à développer une application avec langage C pour simuler le trafic urbain, une file de voiture avec 2 feux de circulation. Donc il nous faut étudier à la fois :

- Le modèle mathématique du mouvement des voitures sur la route
- L'algorithme pour visualiser le modèle ci-dessus
- La réalisation langage C de l'algorithme ci-dessus (Nous avons choisi le technique OpenGL pour le réaliser)

Notre travail, ainsi que ce rapport, est organisé selon ces trois parties.

IDM (Intelligent Driver Model)

IDM est un modèle de la simulation du trafic urbain. Il est développé par Treiber, Hennecke and Helbing en 2000 pour améliorer les modèles existants qui ont moins de caractéristiques réelles.

2.1 Définition

IDM décrit une file de voitures qui se suivent, l'état du trafic en un moment précisé est caractérisé par la position, la vitesse et l'accélération des voitures dans la file. L'accélération d'une voiture, en effet, signifie l'action de conducteur. ça dépend sa vitesse courante, la distance et la différence de vitesse entre elle et la voiture juste de devant.

Les équations décrit ce modèle.

$$\frac{dv}{dt} = a\left[1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s^*}{s}\right)^2\right] \tag{2.1a}$$

$$s^* = s_0 + min\left[0, vT + \frac{v\Delta v}{2\sqrt{ab}}\right] \tag{2.1b}$$

Les explications des paramètres initialisés dans ces équations sont les suivantes :

- v_0 La vitesse désirée sur une route libre.
- T Le temps de sécurité (temps minimum de réaction) quand la voiture suit une autre.
- a Accélération générale. (selon l'habitude du conducteur)
- b Décélération confortable générale.
- s_0 La distance minimum de sécurité entre deux voitures, pare-choc à pare-choc.
- δ L'exposant de l'accélération qui est usuellement 4.

La variable v et s désignent la vitesse courante et la distance de la voiture de devant. Et s^* signifie une distance dynamique désirée.

On peut consédérer la première équation en deux parties : une première partie qui signifie l'accélération désirée $\{a\left[1 - \frac{v}{v_0}\right]^{\delta}\}$ et l'autre partie qui présente la décélération $\frac{s^*}{s}$ provoquée par la baisse de s. Alors, quand l'espace courant s approche s^*, la décélération provoquée va compenser l'accélération désirée, par conséquent, $\frac{dv}{dt}$ va diminuer. D'ailleurs, la décélération va aussi augmenter quand la différence de vitesse augmente fortement vers la voiture devant.

Algorithme de l'application

IDM est un modèle qui est basé sur l'état, c'est-à-dire quand on précise un moment, on peut déterminer les états (position, vitesse...) des voiture, et donc on peut affichier les voiture sur l'écran. Ce que l'on a besoin, c'est seulement une boucle où on mettre à jour les positions des voitures sur l'écran. A partir de cette idée, nous avons établi les structures nécéssaires et l'algorithme principale.

3.1 Structures des données

Les structures de voiture et de la configuration sont définies ci-après :

```
nouvelle structure Config
{
    /** Paramètres pour l'affichage */
     windowWidth:entier;
    windowHeight:entier;
       roadWidth:réel;
        carWidth:réel;
       carLength:réel;

    /** Positions des deux feux sont spécifiées en fonction
      de leurs proportions to the windowWidth*/
                trafficLightPosition1=0.3
                trafficLightPosition2=0.6
    lightChangeDelayWhenSynchronized:entier;
                lightDurationWhenSynchronized:entier;

                /** (Longueur(m) virtuelle de la route/windowWidth(pixel). */
                roadVirtualLengthFactor:réel;

    /** Paramètres fonctionals */
    v0:réel; /**La vitesse désirée(km/h)*/
     T:réel; /**Temps de réaction*/
    s0:réel; /**Gap minimal*/
     a:réel; /**Accélération*/
     b:réel; /**Décélération*/
     moyen:entier; /**Interval moyen du temps d'arrivée en ms*/
};

nouvelle structure Car
{
    x:réel; /**La postion courante*/
    v:réel; /**La vitesse courante*/
    a:réel; /**L'accélération*/
};
```

Ces deux structures sont plutôt ordinaires, ce qui est plus intéressante est la structure de la liste des voirutres. Pour dessiner les voitures sur l'écran, il faut que chacune voiture est une variable de type Car. La question est que le nombre des voitures n'est pas défini, ou peut est très grand. Donc on ne peut bien sûr pas allouer pour chacune voiture un espace mémoire à stocker les donnée. Alors ce que l'on fait, c'est définir une nouvelle structure *CarList* qui est comme une liste circulaire. L'esprit c'est que en fait on s'intéresse seulement aux voitures qui sont actuellement sur l'écran. Quand une voiture sort de l'écran (en ce moment-là la fonction *carOut* sera invoquée), son espace de stockage est prêt pour être réutilisé.

La structure *CarList* :

```
nouvelle structure Car
{
    Car* carArray;//Le tableau des voitures
    int count;//Le nombre des voitures qui sont actuellement sur l'écran
    int size;//La taille du tableau
    int firstCar;//L'index de la première voiture sur l'écran
    int lastCar;//L'index de la dernière voiture sur l'écran
};
```

3.2 Algorithme principal

Les algorithmes que nous avons créés sont décrits au-dessous. Le premier est l'algorithme fondamental, qui décrit le procédure d'exécution de l'application entière.

Algorithme 1 Algorithme fondamental de l'affichage des voitures

Précondition: entrées : config : Config initialisé du fichier de Configuration,
cars : un tableau de Car initialisé

Postcondition: sortie : Une file de voiture simulée sur l'écran

1: **tant-que** vrai **faire** /* La boucle qui dure jusqu'à la fin */
2: Dessiner l'arrière plan
3: **pour tout** car dans le tableau Cars **faire**
4: Dessiner la voiture sur l'écran
5: **fin pour**
6: Mettre à jour les paramètres des voitures présentés par cars
7: **fin tant-que**

Algorithme ci-dessous est un sous-algorithme qui présente le moyen de renouvellement des paramètres des voitures.

Algorithme 2 Sous-algorithme pour renover les paramètres des voitures

Précondition: entrées : cars : un tableau de Car

Postcondition: sortie : Le même tableau des voitures avec les paramètres renouvelés.

pour tout car dans le tableau Cars **faire**
Mettre à jour la position x des voitures
Mettre à jour la vitesse v des voitures
/* C'est la partie la plus importante, qui signifie l'action du conducteur intelligent et qui applique le modèle IDM */
Mettre à jour l'accélération a des voitures
fin pour

L'algorithme suivant est la partie du coeur, qui applique le modèle IDM. C'est un sous-algorithme plus détaillé sur la méthode pour mettre à jour le paramètre de l'accélération.

Algorithme 3 Sous-algorithme pour mettre à jour le paramètre de l'accélération

Précondition: entrées : cars : un tableau de n Car,
config : Config initialisé du fichier de Configuration

Postcondition: sortie : Le même tableau des voitures avec les paramètres d'accélération renouvelés.

pour i de 1 à $n-1$ **faire**

$$car[i].a \leftarrow config.a \left[1 - \left(\frac{car[i].v}{config.v_0}\right)^4 - \left(\frac{config.s_0 + car[i].v * config.T + \frac{car[i].v*(car[i].v - car[i+1].v)}{2\sqrt{config.a * config.b}}}{car[i+1].x - car[i].x}\right)^2\right]$$

fin pour

3.3 Temps d'arrivée des voitures

Pour être plus réel, nous considérons que l'interval entre le temps d'arrivée des deux voitures contiguës doit être aléatoire. L'utilisateur peut fixer un temps moyen dans le fichier de configuration(le nom du paramètre est *moyen*), et l'application va générer un temps aléatoire exponentiel de ce moyen pendant l'exécution.

Notre méthode de génération d'un temps aléatoire unifié à partir du moyen est indiquée par l'équation suivante. L'opérateur *ln* est utilisé pour que la variable générée soit plus unifiée.

$$t = (-moyen * (ln(rand()\%1001 * 0.001f)))\%(2 * moyen) \tag{3.1a}$$

3.4 Synchronisation des feux

Pendant l'exécution de notre application, on peut changer manuellement les deux feux par appuyer sur le clavier 1 et 2. Mais on peut aussi synchroniser les deux feux pour qu'ils puissent changer automatiquement. L'esprit c'est que le délai de changement entre les deux feux doit être bien considérer. Par exemple, quand le premier feu devient vert, une voiture démarre, alors c'est mieux qu'elle touve que le deuxième feu devient aussi vert quand elle arrive. Alors nous somme surpris que ce projet nous donne une chance à reprendre la physique que nous avons apris au lycée.

Le délai de changement entre les deux feux est calculé automatiquement selon :

$$t = \frac{\sqrt{2ad}}{a} \tag{3.2a}$$

L'utilisateur peut aussi fixer ce paramètre à l'aide du fichier de la configuration. Ça concerne le paramètre *lightChangeDelayWhenSynchronized*, qui est initialement -1 qui veut dire **intelligent**.

OpenGL

Après effectuer la première partie de travail sur l'algorithme, nous passons à la suite pour la réalisation de l'application. D'après le but de ce projet, nous choisissons langage C pour la réalisation. Cependant, ce que l'on veut faire, c'est de la programmation graphique, qui ne peut pas être réalisée en utilisant simplement ce que l'on a étudié pendant le cours. Alors nous avons fait de la recherche pour trouver une solution. Enfin nous avons choisi le technique OpenGL pour le faire.

4.1 Introduction

OpenGL(Open Graphics Library) est une librairie multiplate-forme qui définit un ensemble d'API pour faciliter la conception de l'application qui concerne les graphiques 3D/2D. Par rapport à DirectX(l'autre technique similaire sorti par Microsoft), OpenGL supporte plusieurs plate-formes. Par conséquent elle est déjà le standard de l'industrie. Elle contient environ 250 fonctions qui peuvent être utilisées pour affichier des scènes trimentionnelles complex à partir des simples primitives géométriques.

Puisque OpenGL est simplement un ensemble des fonctions, elle seule ne nous permet pas de créer une fenêtre pour montrer notre application. Donc il existe plusieus extensions d'OpenGL qui fournit d'autres fonctionnalités. GLUT(OpenGL Utility Toolkit) est un bon choix.

4.2 Environnement de développement

Pour réaliser cette application, nous avons choisi CodeBlocks comme notre outil de développement. Pour établir l'environnement de OpenGL, nous avons suivi les démarches suivantes :

1. Télécharger et installer CodeBlocks.
2. Télécharger les fichiers de GLUT.
3. Placer glut32.dll à `c:\windows\system`; glut32.lib à `c:\program files\mingw\lib`, et glut.h à `c:\program files\mingw\include\GL`. Ce sont tous les positions par défault, en principe, les fichiers *.dll doit toujours être placé sous le répertoire système Windows; les *.lib et *.h sont mis respectivement sous les répertoire de librairie et de fichier d'en-tête du compilateur.
4. Dans CodeBlocks, créer un nouveau projet GLUT et poursuivre les démarches.Figure4.1
5. Enfin on obtient un projet d'exemple, mais on doit quand même ajouter quelques choses pour le faire marcher. Ajouter `#include<windows.h>` au début du fichier Main.
6. Maintenant on peut déjà lancer cet application d'exemple. Figure4.2

4.3 Structure d'une application OpenGL

Pour maîtriser OpenGL, beaucoup de connaîssances sont nécessaires, mais ici, on peut quand même montrer une structure d'une application simplifiée OpenGL.

La fonction main est généralement comme ça :

```
int main(int argc, char** argv)
{
```

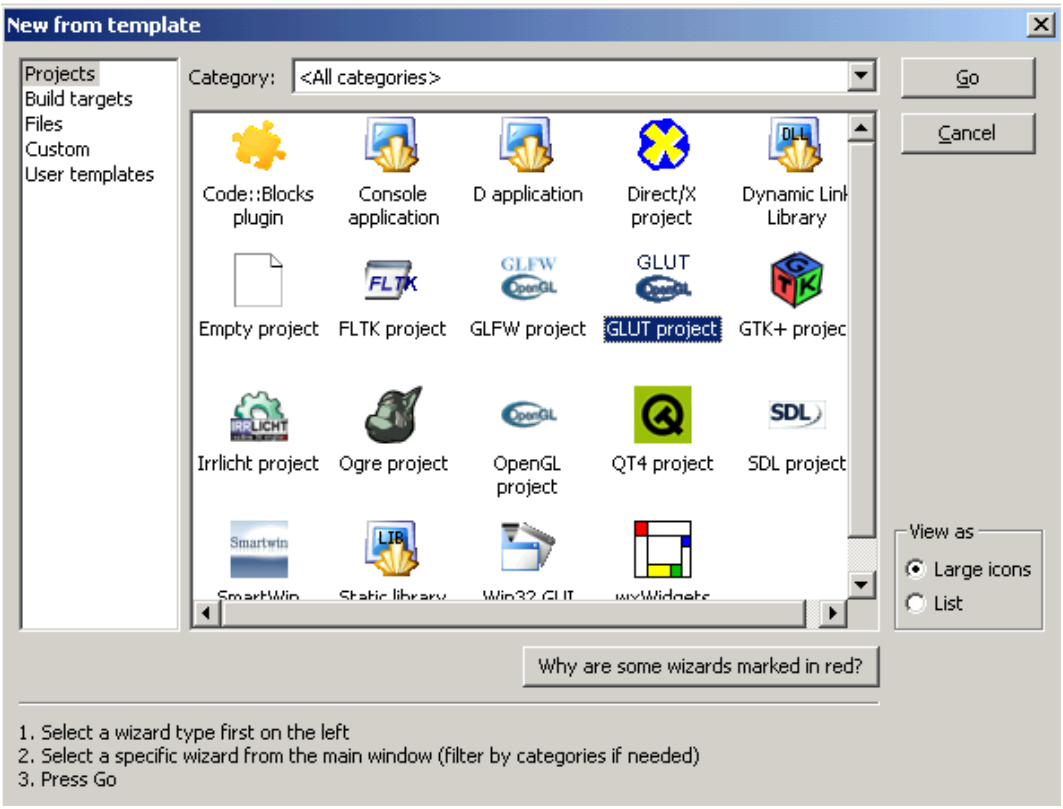

FIGURE 4.1 – Nouveau projet GLUT

```
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE| GLUT_RGB);
glutInitWindowSize(config.windowWidth, config.windowHeight);
glutInitWindowPosition(0,0);
glutCreateWindow(''Traffic Simulation'');
glClearColor(25.0/255,134.0/255,19.0/255,0.3);//Couleur de l'arrière plan
glShadeModel(GL_SMOOTH);
/*Function callback. Fonction myReshape sera invoqué
 *quand la fenêtre change sa forme.*/
glutReshapeFunc(myReshape);
/*Fonction myDisplay actualiser l'écran*/
glutDisplayFunc(myDisplay);
/*mouse est invoqué quand un bouton de la souris est appuyé*/
glutMouseFunc(mouse);

/*Créer un menu*/
glutCreateMenu(menuFonc);
glutAddMenuEntry(''Start'',MENU_START);
glutAddMenuEntry(''Renew configuration'',MENU_RENEW);
glutAddMenuEntry(''Synchronize the traffic lights'',MENU_SYNC);
glutAttachMenu(GLUT_RIGHT_BUTTON);//Lier ce menu avec un bouton

/*Cette dernière ligne commence la boucle principale de cet application*/
glutMainLoop();
```

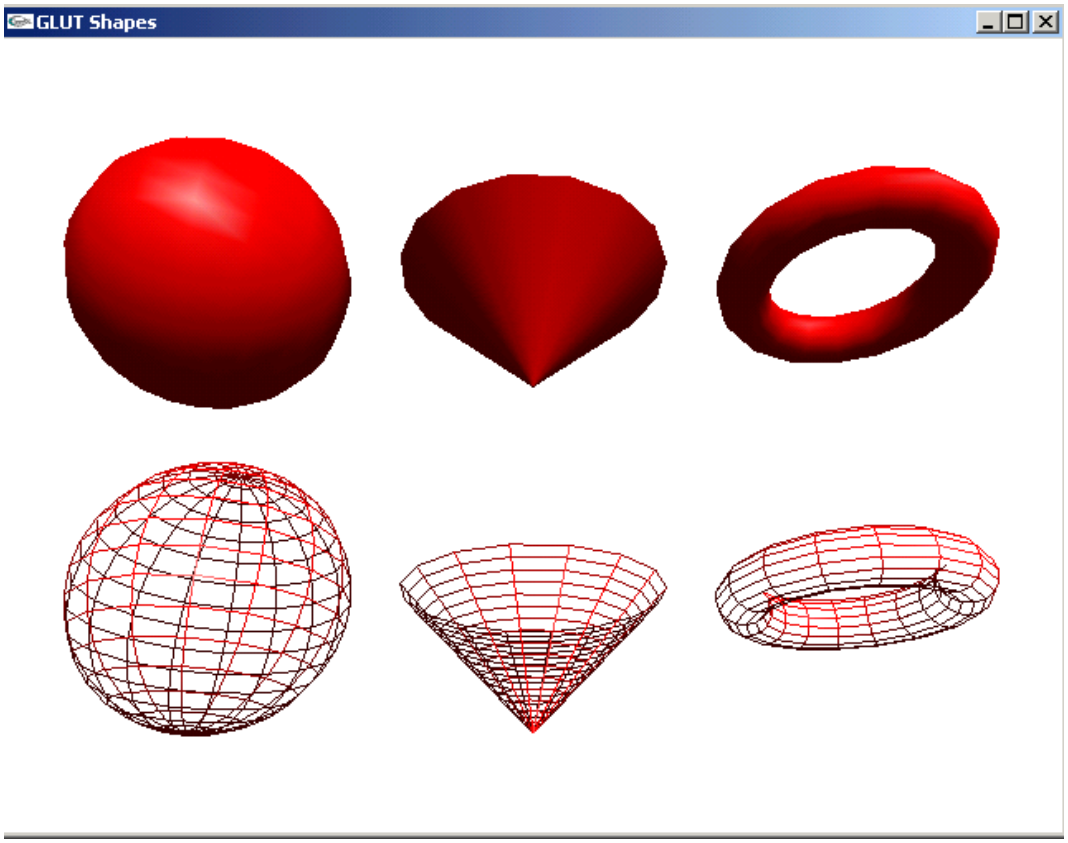

FIGURE 4.2 – Exemple projet GLUT

}

Mise en oeuvre

5.1 Structure des fichiers du projet

Dans le premier temps, nous avons créer une version simplifiée de notre application, simplement pour avoir une impression d'OpenGL. Ensuit, nous avons conçu à nouveau la structure du projet pour ajouter les autres fonctionnalités et aussi pour la clarté.

A la fin, notre projet a une structure comme Figure5.1.

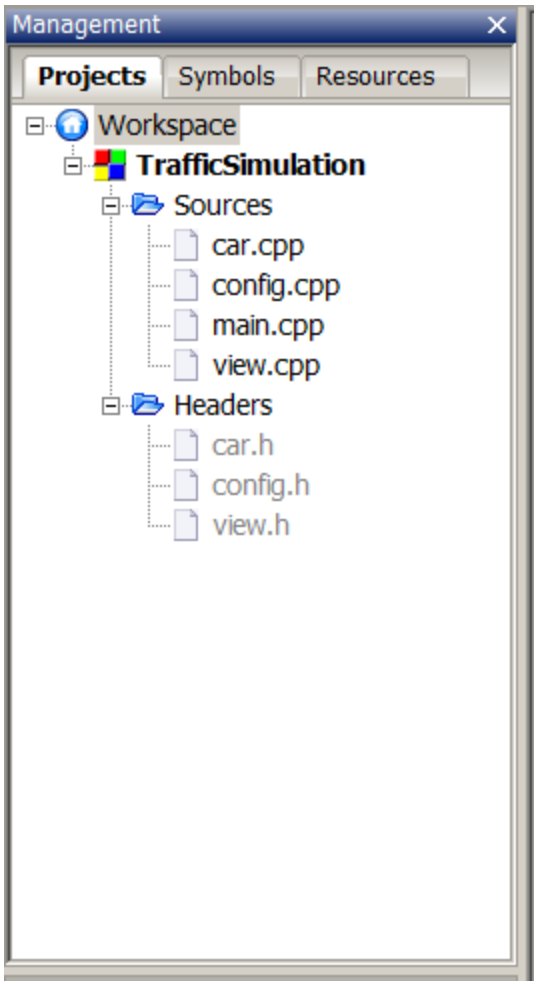

FIGURE 5.1 – Structure des fichiers

Les fonctions des fichiers sont organisés selon l'idée de module. Les explications des fichiers sont présentées ci-après.

config.c(.h)

Il concerne les définitions et les opérations par rapport à la configuration. La fonction principale dans cette partie est la fonction *void initConfigurationFromFile(Config* c)*, qui, comme l'indiquer dans le nom de la fonction, sert à initialiser la configuration de l'application à partir d'un fichier.

car.c(.h)
Il contient les définitions et les opérations qui concernent la structure *Voiture*. Nous avons aussi défini une structure de la liste des voiture, et les autres fonctions telle que *carIn, carOut* correspondent à l'événement d'entrée et de sortie d'une voiture de la fenêtre.

view.c(.h)
Ce fichier contient toutes les fonctions nécéssaires pour l'affichage, y compris les fonctions principales OpenGL. Par exemple, les fonctions indiquées dans la section 4.3, *myResharp, myDisplay, mouse, menuFonc*, etc.

main.c
Le fichier qui contient la fonction *main* qui a une structure typique OpenGL présentée dans le chapitre précédent.

5.2 Interface de l'application

Puisque le but du projet concerne plutôt l'algorithme et le langage C, alors nous n'avons pas attaché beaucoup d'importance sur la beauté de l'interface. Une fois que l'interface peut donner une impression d'une file de voitures qui circule, c'est suffisant.

Alors, voir ci-dessous, l'interface de notre application.

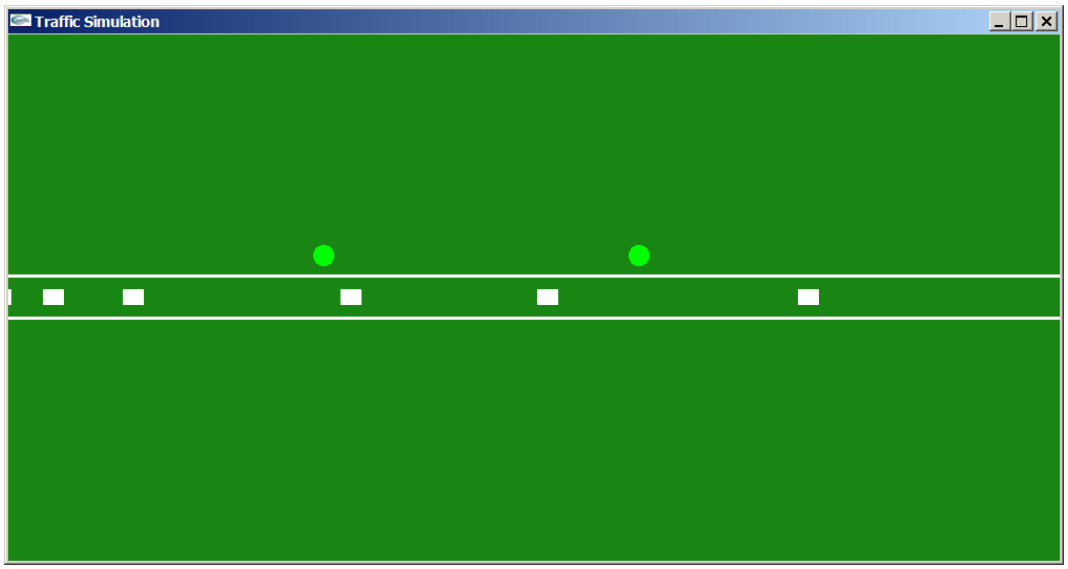

FIGURE 5.2 – Interface de l'application

5.3 Utilisation de l'application

5.3.1 Fichier de configuration

Le fichier de configuration nommé `TS_setting.ini` se trouve sous le répertoire `C:\`. On peut changer les paramètres selons besoins, même pendant l'exécution de l'application. Les paramètres possèdent un nom clair, les significations sont expliquées dans le fichier. L'unité concernant la distance est pixel, et ce qui concerne le temps est milliseconde. Le suivant est le contenu de ce fichier initislisé.

```
#The first two parameters about the window are used for
#the initialisation of the window, so they should not be
# changed during the runtime
```

```
#time unit:ms
windowWidth=1000
windowHeight=500

roadWidth=40
carWidth=15
carLength=20

#The position of traffic lights is specified by their
#proportion to the window width
trafficLightPosition1=0.3
trafficLightPosition2=0.6
lightDurationWhenSynchronized=5000;
lightChangeDelayWhenSynchronized=-1;

#Parameters of the traffic
v0=500
v_begin=50
#unit:s
T=1.5
s0=2
a=30
b=6
#This value should not be too large, because if
#there is no car moving on the screen, openGL will stop refresh it.
moyen=1000
```

5.3.2 Fonctionnalité

Cliquer-droit→ Start
Faire entrer la file de voitures.

Cliquer-droit→ Renew configuration
Renouveler les configurations de l'application après que l'on a changé le fichier de configuration.

Cliquer-droit→ Synchronize the traffic lights
Synchroniser les deux feux.

Appuyer bouton 1
Arrêter la synchronisation des feux et basculer le premier feu.

Appuyer bouton 2
Arrêter la synchronisation des feux et basculer le deuxième feu.

Conclusion

Notre projet concerne à la simulation du trafic. Pour réaliser une application correspondante, nous avons étudié tout d'abord un modèle donné IDM. Ayant compris l'algorithme, nous avons ensuit étudié le technique OpenGL pour le réaliser avec le langage C. Enfin nous avons proposé une application avec assez de fonctionnalité pour simuler une file de voiture.

Avec ce projet, nous obtenons une compréhension plus profonde sur l'algorithme et le langage C, et surtout la méthode pour réaliser un algorithme avec un langage de programmation. Les buts de ce projet sont bien respectés. De plus, nous avons pu avoir, même si pas profondément mais quand même une impression sur la programmation graphique avec OpenGL. En un mot, ce projet est une bonne pratique pour nous.

Simulation du trafic urbain

Département Informatique
3e année
2011 - 2012

Rapport de projet d'Algorithme et de Langage C

Résumé : Le *Modèle de Conducteur Intélligent*(IDM) est un modèle qui représente l'action des conducteurs, c'est-à-dire le mouvement raisonable des voitures dans un trafic urbain. Nous avons établi l'algorithme à partir de ce modèle pour simuler visuellement sur l'écran, une file de voiture sur la route avec 2 feux de circulation. L'application que nous avons réalisé peut être fortement configurable à l'aide du ficher de configuration.

Mots clefs : IDM, Simulation du trafic, Algorithme, OpenGL

Abstract: The *Intelligent Driver Model*(IDM) is a time-continuous car-following model for the simulation of freeway and urban traffic. According to this model, we have proposed an algorithm and then an application to simulate traffic on the screen. This application is developped with C programming language and can be highly custumized with the help of the configuration file.

Keywords: IDM, Traffic simulation, Algorithm, OpenGL

Encadrants
Néron EMMANUEL
emmanuel.neron@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Lei SHANG
lei.shang@etu.univ-tours.fr
Yanxiao HU
yanxiao.hu@etu.univ-tours.fr

DI3 2011 - 2012