



RAPPORT DE STAGE

WEBVIEW

Gagnet Julien

du 01/06/03 au 31/08/03

Table des matières

1	Cadre du stage	5
1.1	Présentation de l'EPFL	5
1.2	Présentation du LSL	5
1.3	Présentation du BIRG	6
1.4	L'entreprise Cyberbotics	6
2	Définition du projet	8
2.1	Cahier des charges	8
2.2	Choix du langage	8
2.3	Outils de programmation	9
2.4	Les 7 versions de l'application	9
3	Sources partagées	11
3.1	La fonction <code>mainloop()</code>	11
3.2	La couche "Environment"	11
3.3	Les Makefile	13
3.4	Affichage OpenGL	15
3.4.1	Lecture et affichage	15
3.4.2	Sélection d'objets	15
3.4.3	Affichage de version	16
3.5	Contrôle de la caméra	17
3.6	Connexions réseaux	17
3.6.1	Connexion TCP	18
3.6.2	Connexion UDP	19
3.6.3	Bascule de connexion	20
4	Windows	21
4.1	Plugin Netscape	21
4.1.1	Le Gecko Sdk	21
4.1.2	Intèraction Navigateur/Plugin	21
4.1.3	Initialisation	22
4.2	Plugin Internet Explorer	24
4.2.1	Utilisation de MingW	24
4.2.2	Chargement de la dll depuis l'ActiveX	24
4.2.3	Cycle de vie de l'ActiveX	25
4.2.4	Chargement depuis une page html	26
4.3	Stand Alone	27
4.3.1	Création d'une fenêtre	27
4.3.2	Icône	28

5	Linux	29
5.1	Plugin Mozilla	29
5.2	Stand Alone	29
6	Mac	31
6.1	Ouverture de fenêtre	31
6.2	Gestion des événements	32
6.3	Création de l'application	32
7	Déploiement sur Internet	33
7.1	Conception	33
7.2	Détection du navigateur	33
7.3	Programme d'installation	34
8	Difficultés rencontrées	35
8.1	L'Async Reply	35
8.2	Des messages qui disparaissent	36
8.3	La Glib pour VisualC++	36
8.4	La Glib pour MacOS	37
8.5	Passage de paramètre à l'ActiveX	37
9	Connaissances retirées	38
9.1	Organisation du travail	38
9.2	Les Makefiles	38
9.3	Programmation des systèmes	39
9.4	Programmation OpenGL	39
9.5	Programmation Réseaux	39
10	Remerciements	40

Table des figures

1	Synthèse des différentes versions	9
2	Exemple d'accès à l'environnement.	12
3	Extrait de la définition de la structure <code>environment</code>	13
4	Détection d'OS pour les déclarations spécifiques dans le Makefile	14
5	En-tête des fonctions manipulant les quaternions.	18
6	Synthèse des échanges clients/serveur.	19
7	Contenu d'une mise à jour	20
8	Utilisation du plugin dans une page Html.	22
9	Mise en place du rendu sous Windows.	23
10	Extrait du chargement d'une dll.	25
11	Utilisation de l'ActiveX dans une page html.	26
12	Vue de l'application sous Windows	27
13	Icône utilisée pour Webview.	28
14	Aperçu du bug d'affichage en deux couleurs.	30
15	Fermeture de fenêtre sous Linux	31
16	Diagramme d'activité de l'installation de Webview.	33
17	Création d'un <code>setup.exe</code> depuis un Makefile.	34
18	Installation du Plugin sous Windows.	35

1 Cadre du stage

Ce stage s'est déroulé à Lausanne en Suisse romande , du 1er juin au 31 août 2003, à l'EPFL (École Polytechniques Fédéral de Lausanne) dans le cadre d'un projet proposé par l'entreprise Cyberbotics, éditrice du logiciel Webots.

1.1 Présentation de l'EPFL

L'EPFL est l'une des deux écoles polytechniques de Suisse, l'autre se trouvant à Zurich (l'EPFZ). Les ingénieurs suisses sont formés principalement dans ces deux écoles prestigieuses ; ainsi l'EPFL compte plus de 6000 élèves (nombre auquel il convient d'ajouter les professeurs, le personnel ...). Elle propose un vaste choix de formations.

L'école est également fortement orientée vers la recherche, là encore dans un nombre impressionnant de laboratoires couvrent des domaines très variés. L'EPFL participe ainsi à de nombreuses publications scientifiques, elle a été impliquée par exemple dans la coupe de l'America, ou dans la réalisation du biowall.

1.2 Présentation du LSL

Ce stage s'est déroulé au sein du LSL (Logic Systems Laboratory), un laboratoire de recherche à la pointe des systèmes logiques. Des études y sont menées dans trois directions principales.

- Les machines de calculs bio inspirées.
- Les systèmes reconfigurables
- Les robots bio-inspirés ¹

La première explore de quelle façon les programmes peuvent évoluer au cours de leur existence et collaborer avec d'autres entités. Les systèmes reconfigurables tel les FPGA sont étudiés également, et des propriétés comme l'auto reconfiguration sont mises en œuvre, ce domaine m'a particulièrement intéressé car il recoupe ce qui m'a été appris pendant ma formation à l'ESEO. La dernière ligne directrice est l'exploration des robots inspirés par la nature, et particulièrement les méthodes de locomotion.

¹En anglais : bio-inspired computing machines, novel reconfigurable systems, biologically inspired robots.

Le LSL s'est illustré dans la réalisation du BioWall, il s'agit d'un mur d'environ 1m de hauteur sur 3m de large,² formé de leds qui sont contrôlées par des milliers de FPGA tous reliés les uns aux autres. Ceci fait de ce mur un formidable outil de travail, et différents type de travaux ont été implémentés, comme par exemple des tissus cellulaires auto réparants, ou encore des simulations d'organismes communicants. Ce mur est tactile, il reconnaît les stimulations des utilisateurs et des applications plus ludiques, comme le jeu de la vie, prennent alors toute leur dimension.

1.3 Présentation du BIRG

J'ai travaillé particulièrement pour le BIRG (Biological Inspired Robotics Group).

Ce groupe de recherche dirigé par Auke Ijspeert travaille dans la robotique, dans le domaine de la locomotion. Ses travaux sont inspirés par des modèles biologiques notamment par celui de la salamandre, animal particulièrement intéressant puisqu'il se déplace aussi bien sur terre que dans l'eau, et qu'il n'a pas évolué depuis des millions d'années.

Le BIRG utilise en particulier le logiciel Webots pour la simulation du robot salamandre. Le plugin que j'ai développé est particulièrement utile pour le groupe puisqu'il permettra de diffuser les travaux de recherche facilement sur internet dans un environnement proche de celui de Webots.

Je dois dire que le cadre a été très stimulant. Le laboratoire explore de nombreux domaines pointus, et j'ai appris énormément en côtoyant différents chercheurs et doctorants. J'ai pu mieux apprécier la dimension du travail de la recherche.

1.4 L'entreprise Cyberbotics

Cyberbotics, qui développe le logiciel Webots, est une petite société constituée par deux salariés. Elle a ses locaux au sein même de l'école, dans une partie réservée aux entreprises, ce qui lui permet de profiter du rayonnement international de l'école et d'être en étroite collaboration avec le LSL.

Webots constitue un puissant outil de simulation de robots, plutôt orienté vers une clientèle professionnelle. Il permet de contrôler toutes les étapes de développement, le design des différents robots en y intégrant des capteurs et actionneurs, la

²Il en existe deux qui peuvent être reliés et former un mur encore plus grand.

définition de leur comportement, la simulation, et le transfert final vers un robots réel.

La simulation est réalisée grâce au moteur physique ODE (Open Dynamic Engine) qui permet un résultat très réaliste. Les forces sont en effet prises en comptes très précisément, les différentes masses et frictions spécifiées. Ce logiciel, très complet, permet entre autre l'importation et l'exportation de scènes au format vrml, mais aussi la création de séquences animées en AVI ou Mpeg.

Ce logiciel existe depuis 7 ans. Il a notamment été utilisé pour la simulation du robot Aibo de Sony. Un concours de robots existe sur Internet, les compétiteurs doivent réaliser un contrôleur de robots qu'ils affronteront au cours de match simulés avec le logiciel Webots. L'intérêt du plugin est alors de pouvoir être plongé dans les matchs, en direct.

2 Définition du projet

Le projet consiste en la réalisation d'une interface accessible depuis Internet, affichant une scène en 3D, où l'utilisateur est totalement immergé et agit directement sur la caméra. Cette scène doit être animée et mise à jour par le réseau.

2.1 Cahier des charges

Les impératifs pour le projet sont nombreux. Un haut niveau de compatibilité est requis, l'utilisateur pouvant aussi bien utiliser Internet Explorer que Netscape ; mais également être sous Windows, Linux ou encore Mac.

Le programme doit pouvoir afficher une scène en 3D complexe, et permettre une forte interactivité. Il est également indispensable qu'il soit facile d'utilisation pour un utilisateur inexpérimenté.

2.2 Choix du langage

Plusieurs possibilités se sont présentées quant au langage à utiliser pour réaliser l'application.

- La première consistait à utiliser un plugin existant, réalisé par une compagnie tierce. Le plus intéressant est sans doute Atmosphere d'Adobe, qui permet d'exploiter l'accélération matérielle pour le rendu 3D. Cependant les possibilités d'interaction ont semblés faibles et l'utilisation de ce plugin rendait le développement dépendant d'une technologie peu répandue.
- La seconde solution était d'écrire le programme en Java, sous forme d'applet. Cette solution était vraisemblablement la plus sûre au niveau de la compatibilité pour les différents systèmes d'exploitations. Cependant, même si Java peut profiter de l'accélération matérielle grâce à Java3D, cette technologie n'est pas accessible depuis un Applet.
- La dernière solution était d'écrire entièrement un plugin pour toutes les plateformes citées et les navigateurs internet qui vont avec. L'intérêt de cette solution était que certaine partie des sources du logiciel Webots pouvaient être adaptées au plugin et rendre le développement plus rapide.

Après réflexion c'est cette dernière solution qui à été choisie.

2.3 Outils de programmation

Le langage de programmation utilisé dans Webots est le C. Celui utilisé par l'api netscape de développement des plugins est le C++. Il est également utilisé pour le développement des controle ActiveX. Il a donc été nécessaire de faire un mélange de ses deux langages.

L'outils de compilation choisi est gcc (ainsi que g++), piloté par un Makefile. L'intérêt de cette approche est qu'elle est assez portable, en effet sous Windows les applications MingW et CygWin permettent d'émuler le fonctionnement du compilateur Gnu. MingW est retenu car il permet de générer des bibliothèques dynamiques (dll) sans autres ajouts, alors que CygWin nécessite l'installation d'une dll particulière qui aurait dû être installées chez chaque client. Les Macs, a partir de la version MacOSX, sont construit sur une architecture Unix et offrent donc l'outils de compilation gcc.

2.4 Les 7 versions de l'application

L'application Webview doit se décliner en de multiples versions. En effet en fonction des restrictions locales, du navigateur utilisé et bien sûr du système d'exploitation, l'utilisateur doit pouvoir choisir la version la plus appropriée, voir Fig. 1.

FIG. 1 – Synthèse des différentes versions

OS	Internet Explorer	Mozilla	Stand Alone
Windows	X	X	X
Linux		X	X
MacosX	X		X

Pour chaque système une version dite "Stand Alone" est prévue. Il s'agit d'une version fonctionnant sans navigateur ; il ne s'agit donc pas d'un plugin mais d'une véritable application qui va ouvrir sa propre fenêtre.

Pour ce qui concerne les navigateurs, il n'y a pas si longtemps, tout le monde était d'accord, les plugins reposaient sur l'api netscape : le Gecko Sdk. Depuis la version 5.5 d'Internet Explorer, ce style de plugin n'est plus supporté,³ au bénéfice de la technologie propriétaire : les Contrôles ActiveX.

³Un controle ActiveX existe permettant de charger les plugins netscape mais cette fonction a été désactivée suite à un problème de sécurité de Microsoft...

On parle d'une version Mozilla, il s'agit en fait du navigateur open source de linux. La version 7 de Netscape est construite sur Mozilla, de même que le navigateur Saphari pour MacOSX⁴. Elle supporte évidemment l'api Netscape.

Le nombre élevé de versions a été l'une des plus grande difficulté, en effet, comme nous le verrons, les librairies, fonctions, api diffèrent. Il a parfois été nécessaire de faire des retour en arrière pour permettre une plus grande portabilité.

⁴Il existe une version d'Internet Explorer 5.5 pour MacOSX et qui supporte uniquement le style des plugins Netscape. Microsoft a d'ailleurs arrêté le développement d'Internet Explorer pour Mac

3 Sources partagées

Développer pour plusieurs systèmes d'exploitation impose l'utilisation d'une couche permettant de faire abstraction de ces systèmes. L'utilisation d'OpenGL pour le rendu simplifie grandement cette tâche puisque la librairie OpenGL est relativement standard. Cependant les fonctions relatives aux réseaux, au système de fichier, et aux threads ne le sont pas. Heureusement Windows s'est inspiré des fonctions Posix dans certains domaines. Les sources partagées sont les différentes fonctions utilisées par toutes les versions de l'application. Il faut noter que le projet s'inscrit dans la structure du logiciel Webots et en utilise également des sources.

3.1 La fonction `mainloop()`

La fonction `mainloop()` est le point de départ pour la section partagée des sources. Cette fonction est lancée dans un thread par la partie spécifique au système d'exploitation.

Deux conditions doivent être vérifiées avant de pouvoir lancer `mainloop` dans un thread. La variable "environment" doit être correctement initialisée et est passée à la fonction en paramètre. L'affichage OpenGL doit être également être déjà initialisé, ceci est fait par la fonction `wglMakeCurrent` pour Windows, `glXMakeCurrent` pour Mac, `glXMakeCurrent` sous Linux. À ce stade, les fonctions standard OpenGL peuvent être appelées sans risque.

L'appel à `mainloop` va alors mettre en place la connection avec le serveur, puis commander l'affichage en boucle de la scène en 3D. Elle appelle également le traitement des messages du système, c'est un fichier `osmsg.c` qui par sa fonction : `readOsMsg(environment* env)` qui va traiter les événements spécifiques au système, principalement lire les déplacements de la souris, écouter les appuis sur les boutons de la souris, ou encore réagir au redimensionnement de la fenêtre.

3.2 La couche "Environment"

Une structure appelée `environment` permet de faire l'abstraction nécessaire par rapport aux entrées de l'utilisateur, ainsi que du système d'affichage. C'est par

cette couche que le système d'exploitation renseignera les variables telles la position et l'état de la souris, ou encore certaine variable moins spécifiques comme l'adresse et le port du serveur auquel se connecter.

L'environnement prend des caractéristiques d'un langage objet en fournissant des accesseurs en lecture ou en écriture pour les différentes variables de la structure. Ceci permet d'utiliser des variables en lecture seule, mais aussi d'effectuer un traitement particulier à l'affectation. Ainsi lorsque la fenêtre prend une nouvelle dimension le flag `dimchange` est levé, voir Fig. 2.

FIG. 2 – Exemple d'accès à l'environnement.

```
void setDimension(int width, int height, environment* env){
    env->width=width;
    env->height=height;
    env->dimchange = true;
}

BOOL isDimChange(environment* env) {
    BOOL state = env->dimchange;
    env->dimchange = false;
    return state;
}
```

L'environnement permet également de connaître l'état de la fenêtre et arrêter la boucle `mainloop` lors d'une demande de fermeture, et permettre ainsi la libération des ressources et connexions réseaux. Ceci est géré par la variable `killme`. Ainsi lorsque le plugin demande la libération des ressources ou lorsque l'utilisateur veut fermer la fenêtre pour la version stand alone, il appellera la méthode :

```
killme(environment * env)
```

puis attendra la fin du thread `mainloop`.

C'est par cette structure que les variables du système d'affichage sont transmises. Il existe en effet pour chaque système deux variables caractérisant généralement le contexte de dessin et la fenêtre :

- pour Windows, un handler sur Window `HWND` ainsi qu'un handler sur le Device Context `hDC`,
- pour MacOSX, un pointeur sur Window `WindowPtr` ainsi que le contexte OpenGL avec un `AGLContext`.

- pour Linux, la fenêtre `Window` et un pointeur sur l'affichage `Display *`

Il en résulte que la structure `environment` n'est pas exactement la même pour les différentes plateformes, mais également du point de vue des accesseurs en lecture et écriture. Ceci est réalisé grâce aux instructions du préprocesseur de type `#ifdef`, voir Fig. 3.

FIG. 3 – Extrait de la définition de la structure `environment`

```
typedef struct {
    ...
    #ifdef WIN32
    HWND window; HDC hdc;
    #else
    #ifdef MACOS
    WindowPtr window; AGLContext context;
    #else
    #else
    Window window; Display * display;
    #endif
    #endif
    ...
}
```

Cette structure alloue toutes les données nécessaires statiquement. En effet la gestion de la mémoire est différente en fonction du système et j'ai eu des problèmes avec l'allocation dynamique de mémoire. Ainsi la variable `environment` est déclarée au début du programme et un pointeur en est généralement passé à toutes les fonctions.

3.3 Les Makefile

Le fichier Makefile est la clé de voûte du projet. En fait, il y en a deux, un pour la version Stand Alone, l'autre pour la version Mozilla.

Ces fichiers sont relativement complexes. Outre les fonctions classiques `make`, `make install` et `make clean`, le fichier détecte le système d'exploitation. Ceci est nécessaire car les appels des différentes bibliothèques diffèrent en fonction de l'OS. Ainsi, l'utilisation de l'API OpenGL se traduit par le passage du paramètre `-lopengl32`

pour Windows, -IGL pour Linux et un -framework OpenGL pour Mac. Les différences sont donc nombreuses et il a été nécessaire de les traiter au cas par cas.

Le fichier Makefile permet également le passage d'arguments au préprocesseur. Ceci est particulièrement utile quand dans les sources même du programme il est nécessaire d'identifier l'OS. Ceci se fait par le passage de définition au compilateur, on aura un -DWINDOWS -DWIN32 pour Windows, un -DUNIX -DLINUX pour Linux et un -DMACOS pour macos. Ceci se fait directement depuis le Makefile par l'identification de l'OS, voir Fig. 4.

FIG. 4 – Détection d'OS pour les déclarations spécifiques dans le Makefile

```
ifeq ($(OSTYPE), windows)
    EXTENSION = .exe
    PLUGIN_DEFINES = -DWINDOWS -DWIN32 -DXP_WIN -D_WINDOWS \
                    -DOS_MSG_LOOP $(VERSION_DEFINE)
    SYSTEM_INCLUDE = -I$(DEPTH)/webots/windows/include \
                    -I$(DEPTH)/webots/windows/include/trash

    LIBS = -lGLU32 -lopengl32 -lgdi32 -lwsock32 -lm
    OBJLIB = $(DEPTH)/webots/windows/lib/glib.a \
            $(DEPTH)/webots/windows/lib/libzlib.a \
endif
```

La version actuelle du programme⁵ est inscrite dans un fichier de configuration lu par le Makefile. Ainsi il est relativement aisé de changer ce numéro de version. ce numéro est envoyé au serveur et un message peut être envoyé en retour si la version du client est dépassée ou bien incompatible.

Le Makefile pour la version windows (plugin) prend également en charge la génération du fichier .res issu du fichier .rc utilisé pour la définition des variables du dll, mais aussi du fichier .def pour ses points d'entrées (les fonctions exportées par la dll). Un autre fichier .rc est également utilisé dans la version stand alone pour spécifier l'icône à utiliser pour l'application.

À la racine du projet un fichier Makefile appelle les autres fichiers Makefiles. La génération des différents fichiers finaux est prise en charge par le Makefile en lui passant le paramètre release, il va alors "stripper" les fichiers exécutables afin d'enlever les différentes informations inutiles, regrouper ces différents fichiers sous forme d'archives tar.gz. Il génère également la création des fichiers setups pour Windows en contrôlant le logiciel InnoSetup.

⁵Trois versions du programmes ont été publiées pendant le stage : la 1.0.0, la 1.1.0 et la 1.1.1

La création d'une nouvelle version release prête pour la diffusion se résume donc à taper `make` pour les trois systèmes d'exploitation et ainsi obtenir dans un répertoire `release` toutes les archives et `setup` prêtes à être installées sur le site web.

3.4 Affichage OpenGL

3.4.1 Lecture et affichage

Les sources permettant la lecture de la scène écrite dans un format propriétaire ⁶, la construction et l'affichage de la scène en OpenGL, ont été directement extraites du logiciel Webots, et je n'ai eu que peu de modifications à effectuer pour les inclure dans le projet Webview.

Ces sources sont principalement axées autour d'une structure formées de nœuds, de primitives standards, d'objets définis par des points de textures (...) ces nœuds permettent de définir les orientations qui seront animées grâce aux informations reçues par la connexion. Un parseur lit le fichier `wbt` et construit l'arbre complet de la scène en mémoire. Il est alors relativement aisé de parcourir la hiérarchie pour retrouver la position d'un nœud par exemple. Le moteur 3D prend alors la structure de la scène pour l'afficher en OpenGL. De nombreuses options sont gérées, comme la gestion de sources d'éclairages multiples l'utilisation de textures ou encore la transparence.

L'intérêt est évident. Le plugin est complètement compatible avec Webots et les modifications qui pourront être apportées à l'affichage ou à la structure seront directement ajoutées au plugin après une recompilation et l'aspect d'une scène sera toujours exactement le même dans Webots que dans Webview. Sans l'apport de ces sources la réalisation aurait sans doute exigé plusieurs mois supplémentaire.

3.4.2 Sélection d'objets

L'utilisateur peut sélectionner un objet directement dans la scène à la souris. La sélection en OpenGL d'un objet s'effectue en plusieurs étapes. Il est nécessaire d'effectuer un rendu de la scène hors écran, dans la zone de la souris. Ce mode de rendu est particulier et il est important de configurer OpenGL avec la commande

⁶Ce format d'extension `.wbt` est très proche du `vrml`. Des identifiants uniques ont été spécialement ajoutés à la hiérarchie pour Webview.

`glRenderMode(GL_SELECT)`, à ce moment il est faut affecter un numéro unique à chaque objet rendu (`glPushName` en OpenGL).

Parmi les objets dessinés au dessus de la souris, on peut sélectionner le plus proche de la caméra et récupérer son numéro. Ceci est possible en utilisant un tableau renseigné par l'OpenGL contenant entre autre la distance et le numéro précédemment associé à l'objet. C'est par la fonction OpenGL :

```
glSelectBuffer(512, buffer);
```

que l'on fournit ce tableau.

Il est alors nécessaire de parcourir la hiérarchie de la scène pour retrouver l'objet associé au numéro retourné. La position de cet objet dans la scène peut alors être transmise au contrôleur de caméra qui en fait sa nouvelle cible.

3.4.3 Affichage de version

L'affichage du numéro de version se fait au lancement de l'application. Plusieurs technique ont été envisagées pour ce faire. Elles sont toutes basées sur l'OpenGL afin de rester au maximum compatible avec les différents systèmes d'exploitation. D'autres solutions plus proche du système aurait pu fonctionner, mais il aurait fallu développer des fonctions indépendantes pour les sept versions de l'application.

- L'affichage de texte sous forme de bitmap, est une technique courante en OpenGL. Une image contenant tous les caractères affichables est construite, son format est nécessairement une puissance de 2 (généralement 256*256). L'avantage de cette méthode est qu'elle permet de contrôler parfaitement la police utilisée et d'être sûr du résultat. Cependant le chargement d'une image par le réseau est une opération plutôt lourde et l'affichage du numéro doit se faire au démarrage de l'application.
- Les lettres peuvent également être codées en dur dans le programme, se sont ainsi directement les pixels de la police qui sont alors renseignés. Cette technique est assez compliquée à mettre en œuvre et le résultat final est de qualité assez médiocre.
- La dernière solution est d'utiliser directement les polices du système. Cette technique est difficile dans la mesure où elle n'est pas portable, tant au niveau des polices employée par le système que pour les fonctions OpenGL utilisée pour le rendu.

C'est cette dernière technique qui à été employée et je me suis donc inspiré de différents tutoriaux affichant du texte en OpenGL. En partant d'une base sous MacOS j'ai fait la part entre les les fonctions propres du système et celle génériques pour adapter la fonction d'affichage de texte aux autres systèmes. Plusieurs essais ont été nécessaires et j'ai notamment pu voir mon texte d'affichage de version en 3D, directement dans la scène.

3.5 Contrôle de la caméra

Une caméra est gérée directement à la souris et permet les actions standards tel que la rotation, le déplacement et le zoom. Le choix des boutons de contrôle a été directement dicté par le logiciel Webots, pour que les utilisateurs de Webots puissent prendre le plugin en main facilement. Des aménagements ont dû être réalisés pour les utilisateurs de Mac qui, dans le pire des cas, ont une souris avec un boutons unique, la technique employées est celle de la combinaison de touche avec la souris. Ainsi la touche Ctrl simule le bouton droit, Shift le bouton du milieu.

Les fonctions de déplacements et zoom de la caméra sont relativement simples. La caméra est représentée par un référentiel constitué des vecteurs \vec{i} , \vec{j} , \vec{k} avec \vec{k} pointant dans la direction du regard, il suffit en effet de translater le référentiel de la caméra selon ses différents axes, \vec{k} pour le zoom, \vec{i} et \vec{j} pour le déplacements, le tout proportionnellement aux déplacement de la souris.

Les fonctions de rotation ont été réalisées grâce à la mathématique des quaternions.⁷ Celle-ci est en effet extrêmement utile lorsqu'il s'agit d'effectuer des rotations sur des référentiels, voir Fig. 5. Une fonction permet de passer de la notation vecteur/angle couramment employée en OpenGL ou en vrml à celle du référentiel de la caméra et inversement. La rotation s'effectue alors par rapport au centre d'intérêt (l'objet sélectionné) sinon l'origine du monde.

3.6 Connexions réseaux

La scène est en permanence mise à jour depuis Internet, et ce, en utilisant les deux protocoles : TCP et UDP.

⁷Une première version à été implémentée par des fonctions simples de rotation dans les différents plans mais les résultats n'étaient pas satisfaisants.

FIG. 5 – En-tête des fonctions manipulant les quaternions.

```

// Définition d'un vecteur et d'un référentiel
typedef struct{ float x,z,y; } vector;
typedef struct{ vector 0,i,j,k; } referentiel;

// Définition d'un quaternion
typedef struct {
    float s;
    vector v;
} quaternion;

// transforme un vecteur en quaternion
quaternion v2q( vector v );
// transforme une rotation vecteur/angle en quaternion
quaternion getQRot( vector axis, float angle);
// l'inversion d'un quaternion :  $q^{-1}$ 
quaternion invQ( quaternion q );
// la multiplication de deux quaternions
quaternion mult_quaternion(quaternion q1, quaternion q2);
// la multiplication de trois quaternions
quaternion mult_quaternion3(quaternion q1, quaternion q2,
                             quaternion q3);
// la rotation d'un référentiel par un quaternion
void rotateReferentiel( referentiel * r, quaternion q );

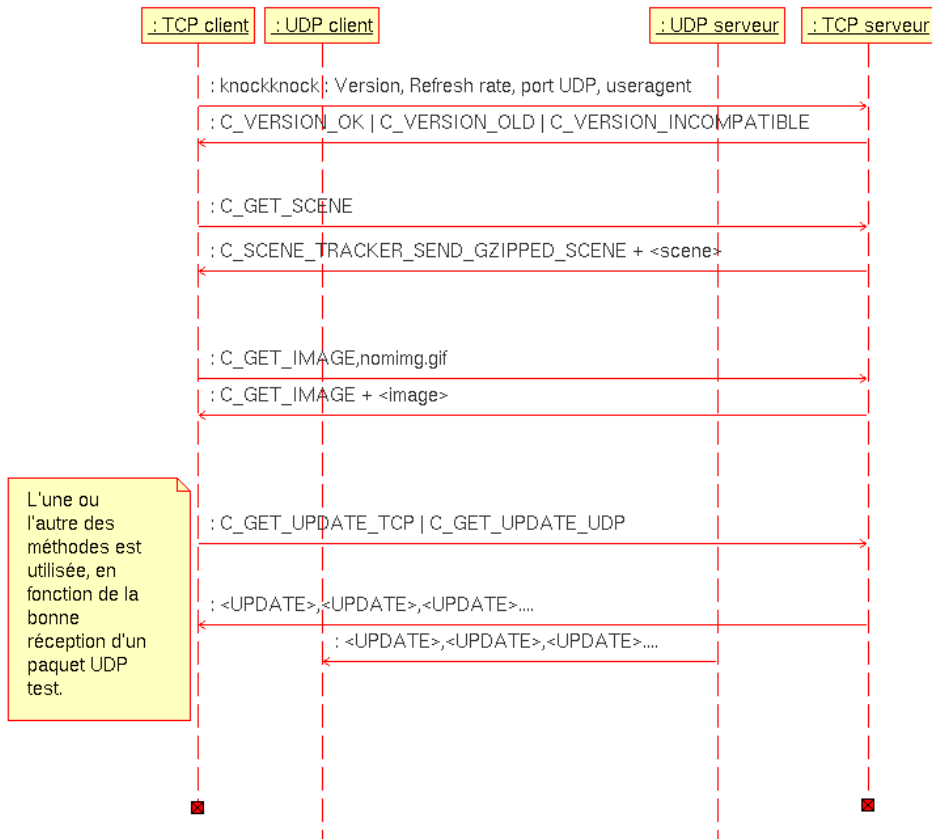
```

3.6.1 Connexion TCP

Le protocole TCP fonctionne en mode connecté. Les données sont toutes transmises et arrivent dans l'ordre. Ces garanties sont nécessaires pour l'envoi de la structure de la scène, et celle des images.

Un certain nombre d'opérations sont nécessaires avant l'attente de mise à jour, voir Fig. 6. Le premier message envoyé est une présentation du client auprès du serveur, le knock-knock. Le serveur pourrait notifier au client si une nouvelle version est disponible. Le client demande la scène, alors le serveur la lui transmet ; le fichier est transmis au format zip puis est directement décompressé dans le flux. Des éventuelles textures peuvent alors être téléchargées pour les différents objets. Ceci fait, le client va demander des mises à jour.

FIG. 6 – Synthèse des échanges clients/serveur.



3.6.2 Connexion UDP

Les envois de mise à jour sont réalisés grâce au protocole UDP, beaucoup plus rapide que le TCP. Il ne garanti cependant pas la bonne transmission des messages, ni leur ordre d'arrivée.

Ce protocole est donc particulièrement approprié pour le temps réel, où il est plus important d'avoir un rendu fluide. Si un paquet se perd il sera périmé avant d'avoir pu être renvoyé. Pour s'assurer l'ordre d'arrivée les paquets sont numérotés, le plus récent est toujours conservé, les paquets plus ancien seront ignorés.

Un paquet UDP contient toutes les informations concernant les noeuds en mouvement. Un noeud est ainsi repéré dans la hiérarchie par trois coordonnées spatiales, et quatre coordonnées d'orientation (un axe et un angle) ; soit 7 doubles auxquels il faut ajouter l'identifiant de l'objet. L'en-tête du paquet contient le numéro d'ordre et le nombre de noeuds qui sont mis à jour.

FIG. 7 – Contenu d'une mise à jour

Paquet UDP :

id=23	count_updt=3	upate 1	upate 2	upate 3
-------	--------------	---------	---------	---------

Mise à jour (update) :

unique_id	x	y	z	axe x	axe y	axe z	angle
-----------	---	---	---	-------	-------	-------	-------

3.6.3 Bascule de connexion

Le protocole UDP est très performant, cependant un problème est rapidement apparu. Dans certain cas les paquets UDP n'arrivaient pas jusqu'au client. Ceci se produit généralement pour des modems ADSL qui effectuent du NAT.⁸

La solution consiste à détecter dans un premier temps le bon fonctionnement du protocole UDP. Le serveur va envoyer quelques paquets UDP après la réception du message de présentation. Ainsi le client pourra envoyer soit le message :

C_WEBVIEW_GET_UPDATE_TCP, soit

C_WEBVIEW_GET_UPDATE_UDP,

en fonction de la bonne réception des paquets de test.

La réception des mises à jour en mode TCP est très proche de l'UDP. Le format des paquets de mise à jour est le même, la numérotation est conservée pour limiter les modifications du protocole. Pendant les tests sur un réseau local, de légers saccadements ont pu être observés dans ce mode de connexion.

⁸Network Address Translation, utilisé pour que plusieurs ordinateurs puissent utiliser une connexion Internet avec la même adresse IP.

4 Windows

4.1 Plugin Netscape

4.1.1 Le Gecko Sdk

L'écriture d'un plugin pour Netscape passe par l'utilisation du Gecko Sdk. Il s'agit d'une api fournissant les méthodes par lesquelles le navigateur communique avec le plugin. Ce n'est finalement qu'une librairie dynamique (une dll). Il s'agit donc d'implémenter un certain nombre de méthodes pour espérer obtenir un plugin chargé par netscape.

Cette partie du développement était nécessairement en C++, le reste de l'application étant lui écrit en C. Des aménagements ont dû être réalisés pour pouvoir les inclure, en particulier grâce l'utilisation de l'instruction `extern "C"` dans les en-têtes des fichiers C.

Le plugin doit ainsi implémenter une méthode d'ouverture `NS_NewPluginInstance`, une méthode de fermeture `NS_DestroyPluginInstance`, ainsi qu'un constructeur et un destructeur de l'objet `nsPluginInstanceBase`.

En plus des ces fonctions minimales attendue (ouverture, fermeture...) le Gecko Sdk fournit des méthodes génériques de lecture par réseaux ou encore d'allocation de la mémoire, l'intérêt de ces méthodes est d'être indépendantes du système. Cependant l'utilisation de telles fonctions s'est vite révélée d'un intérêt limité puisque les version stand alone n'utilisent pas cette api.

4.1.2 Intèraction Navigateur/Plugin

Le type Mime du plugin doit être renseigné par l'implémentation de la méthode `NPP_GetMIMEDescription`, ce type est `application/x-webots`, c'est une application de type webots, et les caractères "x-" signifie que le type n'est pas enregistré.⁹ Plus tard, dans la page html, c'est l'utilisation de ce type mime qui indique au navigateur de charger le plugin, voir Fig. 8. Ce fichier dll se situe dans le répertoire plugin et doit nécessairement commencer par np... (ici `npwebview.dll`); il fallait le savoir.

⁹Un document RFC (Request For Comment) doit être établi, la procédure est décrite dans la RFC-2048.

En retour, le navigateur fournit sa variable d'identification du navigateur appelée `UserAgent`, par exemple :

```
Mozilla/5.0 (compatible; Netscape/3;Windows)
```

C'est une information relativement importante, surtout lorsqu'il s'agit de régler des bugs ou d'établir des statistiques. Cette variable est donc passée au serveur à l'ouverture de la session.

Le navigateur fournit également la liste des arguments qui sont passés au plugin par la page html via la balise `embed`, voir Fig. 8 Les attributs `width` et `height` sont standards et sont directement traités comme du code html.¹⁰ Cette liste est fournie par la structure `nsPluginCreateData` qui contient le tableau `argc` qu'il convient de traiter. L'information ici transmise au plugin est l'adresse du serveur, le port de connexion TCP et le nom de la scène ; ceci grâce à la variable `scene`.

FIG. 8 – Utilisation du plugin dans une page Html.

```
<embed type="application/x-webots"
      width="100%" height="100%"
      scene="wtp ://cyberboticspcl.epfl.ch :10014/new.wbt">
```

4.1.3 Initialisation

Toutes les initialisations sont réalisées dans la méthode `NS_NewPluginInstance`, la variable "environment" est alors créée et initialisée grâce à la structure `srcns-PluginCreateData` qui contient les paramètres vus précédemment.

Il est alors nécessaire d'activer l'affichage OpenGL. La méthode :

```
init(NPWindow* aWindow){...
```

est appelée juste avant l'affichage du plugin, son argument est de type `NPWindow` qui est défini par l'api Gecko, il s'agit en fait de la variable propre au système d'exploitation. Ainsi, sous Windows la structure est "typée" pour obtenir le `HWND` nécessaire :

```
hWnd = (HWND)aWindow->window;
```

C'est à partir de ce `HWND` qu'une série de fonctions va mettre en place l'affichage OpenGL, voir Fig. 9 où tous les différents tests de retour ont été supprimés pour

¹⁰À noter qu'il n'est pas possible pour un controle ActiveX de spécifier une dimension en pourcentage lorsqu'il est inséré dans un tableau.

faciliter la lecture. Il s'agit de paramétrer l'affichage et le mode de rendu au moyen d'un `PIXELFORMATDESCRIPTOR`, puis de créer un contexte de rendu ; ces fonctions ne peuvent pas être directement appelées dans cette fonction `init` et un thread doit prendre la relève. Une fois la méthode `wglMakeCurrent` appelée avec succès, le travail est fini, et la fonction `mainloop(...)` est appelée.

FIG. 9 – Mise en place du rendu sous Windows.

```
// paramètres du rendu
static PIXELFORMATDESCRIPTOR pfd={
    sizeof(PIXELFORMATDESCRIPTOR),1,
    PFD_DRAW_TO_WINDOW|PFD_SUPPORT_OPENGL|PFD_DOUBLEBUFFER,
    PFD_TYPE_RGBA,16, // profondeur des couleurs
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    16, // 16 bits Z-buffer
    0,0,
    PFD_MAIN_PLANE,
    0,0,0,0
};

// variables d'affichage
HDC hDC;
static HGLRC context;
GLuint PixelFormat;

// mise en place du rendu
hDC=GetDC(hWnd);
PixelFormat=ChoosePixelFormat(hDC,&pfd);
SetPixelFormat(hDC,PixelFormat,&pfd);
context=wglCreateContext(hDC);
wglMakeCurrent(hDC,context);
```

Je dois dire que l'exemple fournit par le Gecko Sdk m'a beaucoup aidé. Cependant il s'agissait d'un projet complètement fonctionnel pour MSVisualC++ que j'ai dû adapter pour MingW en essayant faire le lien entre le fichier Makefile généré par MSVC++ et celui traditionnel qu'utilise MingW. En partant de l'exemple de base affichant un simple texte à l'écran (la variable `UserAgent`), j'ai d'abord testé l'affichage d'une scène rudimentaire et son animation, pour finalement appeler le `render` spécifique à Webots.

4.2 Plugin Internet Explorer

4.2.1 Utilisation de MingW

Les plugins pour Internet Explorer sont différents de ceux employés dans Netscape, un plugin est un contrôle ActiveX. Cette particularité fut une difficulté du projet.

J'ai dans un premier temps facilement pu réaliser un ActiveX assez basique en utilisant un exemple de projet fournis par MSVisualC++, cependant l'utilisation de ce compilateur pour le projet posait problème, elle aurait obligé à maintenir plusieurs versions des sources et utiliser plusieurs outils pour générer les fichiers finaux. C'est pourquoi j'ai tenté de convertir l'exemple vers l'outil MingW, comme pour l'exemple du plugin, en convertissant le pseudo Makefile de MSVC++. Cependant cela s'est vite révélé impossible.

Un ActiveX est un fichier d'extension ocx. Comme pour une dll, elle exporte certaines fonctions et est chargée dynamiquement. D'ailleurs sur un forum, à la question "*Quelle est la différence entre un ocx et une dll ?*" quelqu'un répondait "*L'extension !*".

Malheureusement ce n'est pas la seule différence, un ActiveX doit en plus exporter des fonctions requises :

```
DllGetClassObject  
DllCanUnloadNow  
DllRegisterServer  
DllUnregisterServer
```

Ces fonctions sont nécessaires à l'enregistrement du contrôle auprès du système, et leur implémentation nécessite l'utilisation de bibliothèques propriétaires, et donc non accessibles depuis MingW. Il fallait donc utiliser MSVC++ pour le développement. C'était pourtant sans compter avec l'impossibilité d'utiliser les fonctions de la Glib avec le compilateur de Microsoft (voir la section 8.3 à ce sujet).

4.2.2 Chargement de la dll depuis l'ActiveX

La solution qui a alors été employée consiste à charger dynamiquement le plugin précédemment développé pour netscape, et utiliser l'ActiveX uniquement pour le chargement et la libération de la dll.

La dll est modifiée pour permettre un point d'entrée et un point de sortie, elle exporte ainsi deux nouvelles méthodes :

```
void OSCALL StartFromActiveX(HWND hWnd)
void OSCALL StopFromActiveX()
```

La première fournit le `HWND`, qui nous l'avons vu constitue la pièce principale pour la mise en place de l'affichage OpenGL, elle va donc appeler les mêmes fonctions d'initialisation puis laisser la place à la fonction `mainloop()`. La seconde demande à la fonction `mainloop()` de se terminer via la méthode `killme()` de l'environnement qui entraîne l'arrêt des threads de communication.

Le chargement de la dll s'effectue en deux étapes. Dans un premier temps l'ActiveX va chercher son propre chemin, pour en déduire le chemin de la dll et alors la charger. Les deux méthodes exportées peuvent alors être recherchées dans la dll au moyen de la méthode `GetProcAddress(...)` ; voir la Fig. 10 où les tests de retours des fonctions ont été supprimés.

FIG. 10 – Extrait du chargement d'une dll.

```
// recherche du chemin du plugin
HMODULE hm = GetModuleHandle("webview.ocx");
char st[MAX_PATH];
GetModuleFileName(hm, st, MAX_PATH);
char *ti = strrchr(st, '\\')+1;
*ti = '\\0';
strcat(st, "npwebview.dll");

// chargement des fonctions exportées
hLib = LoadLibrary(st);
ProcStart=(StartFromActiveX)GetProcAddress(hLib,"StartFromActiveX");
ProcStop =(StopFromActiveX) GetProcAddress(hLib,"StopFromActiveX" );
```

4.2.3 Cycle de vie de l'ActiveX

Comme pour l'api Gecko un certain nombre de méthodes rythment le cycle de vie de l'ActiveX. Le constructeur permet le chargement de la dll comme vu dans la section précédente alors que le destructeur libère la librairie avec un `FreeLibrairie(...)`.

L'appel à la méthode `StartFromActiveX` se fait pendant l'appel à `OnDraw(...)` du contrôle. En effet elle fournit la variable `pdc` de type `CDC*` qui contient toute les informations utiles au dessin et encapsule surtout notre `HWND` indispensable au rendu.

En fin de ce cycle on trouve la méthode `OnFinalRelease()`, chargée de demander l'arrêt du plugin en appelant sa méthode `StopFromActiveX`. Il ne faut pas négliger la méthode `OnResetState()`, elle est appelée par exemple lorsque l'utilisateur appuie sur le bouton "rafraichir" de son navigateur. Il est alors important d'appeler ici également la méthode `StopFromActiveX`, puis d'attendre le prochain `OnDraw(...)` pour relancer le processus.

4.2.4 Chargement depuis une page html

La demande de chargement est légèrement différente par rapport au plugin de Netscape, il ne s'agit plus de la balise `embed`, mais de la balise `object`, voir Fig. 11. Un ActiveX est plus qu'un simple plugin pour navigateur, il s'agit en fait d'un *Serveur COM*. Il implémente par exemple des fonctions de persistance (nécessaire pour conserver l'état de l'objet entre plusieurs utilisation). Cet Objet doit être enregistré auprès du système avant de pouvoir l'utiliser. L'ActiveX est enregistré au moyen d'un identifiant unique, le `CLSID`. Il n'est donc pas nécessaire d'installer le plugin dans un répertoire particulier pour pouvoir l'utiliser, simplement de l'enregistrer auprès du système (ceci se fait grâce à la commande `regsvr32`). C'est le paramètre `classid` de la balise `html Object` qui permet de spécifier à Internet Explorer quel ActiveX charger.

FIG. 11 – Utilisation de l'ActiveX dans une page html.

```
<object classid="CLSID:7929103C-B5B0-4EFD-95D9-5C0E262D6B0A"  
        height="80%" width="80%">  
</object>
```

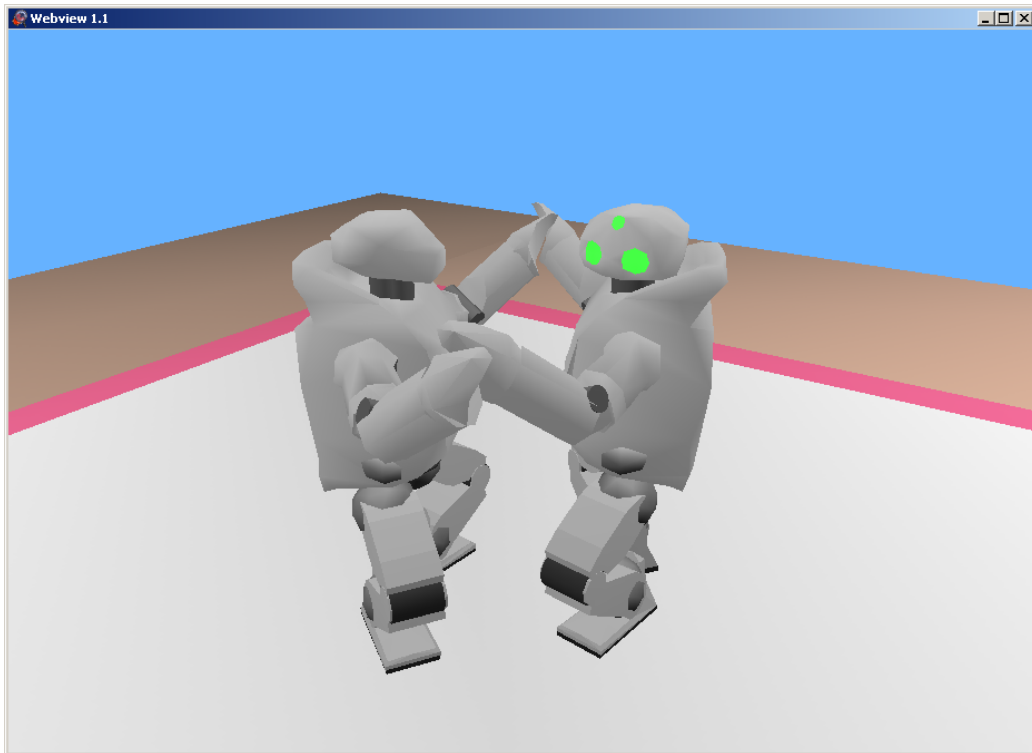
4.3 Stand Alone

4.3.1 Création d'une fenêtre

La version Stand Alone pour Windows constitue une version directement exécutable (ie un .exe), elle se passe donc d'un navigateur. La création d'une fenêtre est donc la nouveauté.

C'est une opération assez courante, il s'agit de renseigner un certain nombre de paramètres (titre, taille...) dans une structure WNDCLASS puis d'appeler la fonction `CreateWindowEx`; et surtout de ne pas oublier de spécifier une fonction qui traitera les messages du système. La Fig. 12 est une capture d'écran de cette version stand alone.

FIG. 12 – Vue de l'application sous Windows



4.3.2 Icône

Une icône a été réalisée pour l'exécutable, voir Fig. 13, je me suis largement inspiré de celle de Webots, une coccinelle. Une icône supporte plusieurs résolutions, l'icône créée en possède deux 32*32 pxl et 48*48 pxl, et la transparence est utilisée. MSVisualC++ permet de convertir assez rapidement des images en .ico. C'est le fichier .rc compiler en .res qui spécifie l'icône à utiliser :

```
2302    ICON    DISCARDABLE    "../icon/webview.ico"
```

Un numéro arbitraire lui est attribué, il est utilisé dans l'application pour changer également l'icône de la fenêtre, pendant sa création :

```
LoadImage(GetModuleHandle(NULL), MAKEINTRESOURCE(2302),  
          IMAGE_ICON, 16, 16, 0);
```

FIG. 13 – Icône utilisée pour Webview.



5 Linux

Comme vu au précédemment deux versions ont été développées pour Linux, une version plugin et stand alone.

5.1 Plugin Mozilla

J'ai rencontré beaucoup de problèmes pour la version plugin de Webview, sous Linux. Le principe reste le même que pour la version Netscape pour Windows, implémenter le Gecko Sdk ; les différences tiennent principalement au système de fenêtre et à la gestion des événements.

Le fonctionnement est donc similaire il s'agit de renseigner le type MIME, de récupérer la variable User Agent, puis de mettre en place l'affichage OpenGL. Cependant la nécessité de synchronisation a été un vrai problème, voir difficultés rencontrées, page 35, et qui a été résolu grâce à l'étude d'un autre plugin, celui du logiciel de modélisation 3D : Blender.

Pendant la dernière semaine du stage, un autre problème est survenu, encore plus incompréhensible. L'affichage n'apparaissait plus qu'en deux couleurs, voir Fig 14. Après avoir passé tout le code source en revue et réfléchi aux modifications qui auraient pu être apportées pendant le développement sur d'autres plateformes, j'ai finalement retesté des archives du plugin beaucoup plus ancienne et qui marchaient pour m'apercevoir que le problème apparaissait également. Ce problème a donc été attribué à la configuration logiciel, et n'a pas été élucidé.

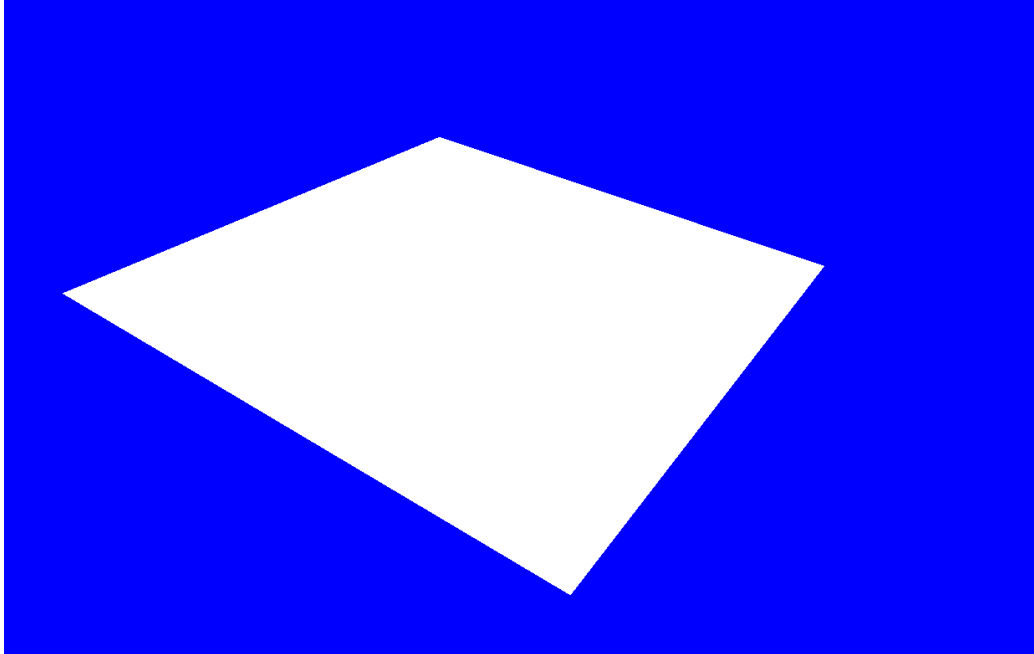
5.2 Stand Alone

La version stand alone, n'a pas posé de réel souci. Elle a permis de mieux comprendre le problème du bug "Async Reply" (voir section 8.1, puisqu'il n'y est jamais apparu.

Comme pour la version Windows, il s'agit d'ouvrir une fenêtre, d'y établir l'affichage OpenGL, puis de traiter les différents messages. Un petit défaut est alors apparu : lors de la fermeture de la fenêtre le message suivant apparaissait dans la console :

```
X connection to :0.0 broken (explicit kill or server shutdown).
```

FIG. 14 – Aperçu du bug d’affichage en deux couleurs.



Se problème se règle à un niveau assez bas, en spécifiant au Window Manager un type d’événement `WM_DELETE_WINDOW`, voir Fig 15.

FIG. 15 – Fermeture de fenêtre sous Linux

```
Atom WM_DELETE_WINDOW=XInternAtom(getDisplay(env),
                                   "WM_DELETE_WINDOW", True);
static BOOL xkillinit = false;
if (!xkillinit){
    xkillinit = true;
    XSetWMProtocols(getDisplay(env), getWindow(env),
                    &WM_DELETE_WINDOW, 1);
}

while ((nbevent=XPending(getDisplay(env)))>0) {
    XNextEvent(getDisplay(env), xevent);

    if (xevent->xany.type==ClientMessage
        && xevent->xclient.data.l[0]==WM_DELETE_WINDOW)
        killme(env);
    ...
}
```

6 Mac

Depuis MacOSX, le système d'exploitation repose sur un architecture Unix, et le développement d'une application peut se faire en utilisant le compilateur gcc associé au Makefile. Seule une version stand alone à pu être réalisée pendant le stage, pas la version plugin. Ceci est dû à certaines difficultés techniques liées au compilateur et au manque de temps.

6.1 Ouverture de fenêtre

Comme pour Windows et Linux il s'agit dans un premier temps d'ouvrir une fenêtre. Ceci est réalisé par l'appel de la fonction `NewCWindow` avec tous ses paramètres correctement renseignés. Le nom qui est donnée à la fenêtre doit être passé avec la notation des chaînes de caractères Pascal, la taille de la chaîne de caractère en première position, comme ceci :

```
const char *title = "\pWebview";
```

6.2 Gestion des événements

La gestion des événements pour Mac est particulière. En plus des fonctions standard de gestion de la souris, il est nécessaire de gérer les déplacements de la fenêtre lorsque l'utilisateur clique sur la barre de déplacement, ou encore la réduction de la fenêtre. Cette particularité rend le développement plus difficile, et toute la documentation en ligne de Mac est basée sur des exemples écrits en Pascal qu'il faut donc adapter.

6.3 Création de l'application

La compilation se fait grâce à l'utilisation du Makefile standard. Cependant les différences sont importantes quant à la déclaration des bibliothèques. On aura ainsi comme paramètres :

`-framework System -framework OpenGL -framework GLUT`

`-framework AGL -framework ApplicationServices -framework Carbon`

Mac utilise un précompilateur qui bloque la bonne compilation de certaines parties des sources, il est désactivé en utilisant la commande `-no-cpp-precomp` dans le Makefile.

L'application finale est contenue dans un répertoire dont le nom finit par `.app`, le Finder du Mac l'interprétera comme un fichier exécutable. Le contenu du répertoire "webview.app" donc est bien particulier, il contient un seul répertoire "Contents" qui contient le fichier "Info.plist" décrivant le contenu de l'application : le nom du fichier exécutable et l'icône à utiliser principalement.

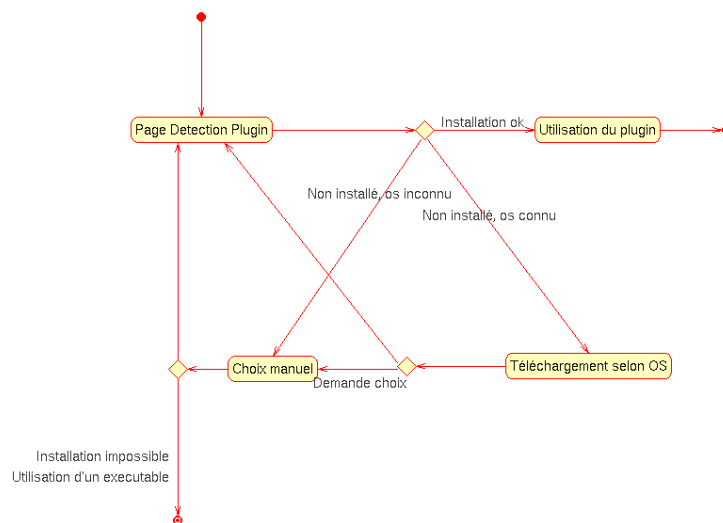
7 Déploiement sur Internet

Quelques pages Internet ont été réalisées visant à utiliser plus facilement le plugin, installation et utilisation.

7.1 Conception

Une modélisation UML à permis de mieux évaluer le fonctionnement des pages web. C'est un diagramme d'activité qui reprend toutes les étapes intervenant avant l'utilisation du Plugin. Dans un premier temps, une détection automatique du navigateur et du système tente d'orienter l'utilisateur vers la version du plugin la plus adaptés. Si ce choix ne lui convient pas il peut alors procéder à une installation manuelle du plugin ou bien de la version stand alone. Voir le diagramme, Fig 16

FIG. 16 – Diagramme d'activité de l'installation de Webview.



7.2 Détection du navigateur

La détection du navigateur est indispensable pour proposer l'installation la plus adaptée, mais également pour appeler soit l'activex, soit un plugin au style Netscape.

Cette détection utilise dans un premier temps un script javascript pour détecter le système d'exploitation et le navigateur utilisé ; c'est la variable `useragent` déjà évoquée qui est traitée. Ces deux variables sont alors passées en argument à une page en php (`redirect.php`) qui va adapter au mieux la page de téléchargement, ou rediriger vers les choix d'installation manuelle.

7.3 Programme d'installation

Pour windows un fichier de type `setup.exe` est utilisé pour les installations de l'ActiveX, du plugin Netscape et de la stand alone, voir Fig. 18 C'est un logiciel de création d'installateur qui est utilisé, InnoSetup. Un incroyable outil qui va permettre à l'utilisateur de choisir le répertoire d'installation, enregistrer l'ActiveX auprès du système, et générer des programmes de désinstallation ; ajoutons qu'il est gratuit. C'est un script qui permet de spécifier les différents textes à afficher pendant l'installation, et de paramétrer l'installation. InnoSetup peut être appelé depuis la ligne de commande et générer à la demande les fichiers `setups` finaux, malheureusement ceci ne peut être fait directement depuis le Makefile sous MingW ; la solution est de générer un script windows (un `.bat`) qui appellera InnoSetup, appelé par MingW, voir Fig. 17.

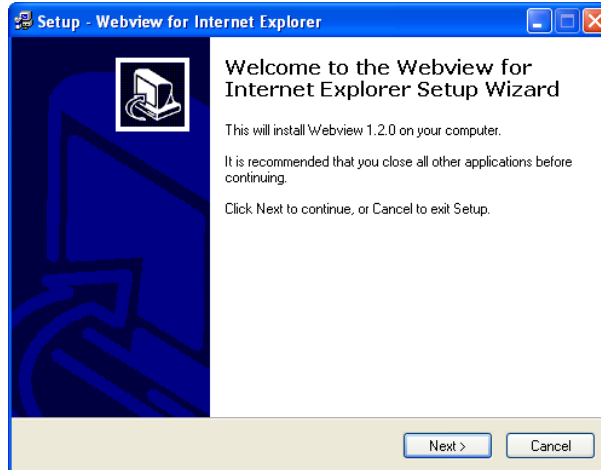
FIG. 17 – Création d'un `setup.exe` depuis un Makefile.

```
# $(TMPMKSETUP) est le script temporaire windows, .bat
# $(INNO) est le programme de création de setup
echo '$(INNO) /cc $(SETUP_PATH)/setup_ie/setup.iss' > $(TMPMKSETUP)
cmd /c $(TMPMKSETUP)
```

Pour Linux les fichiers executables sont réunis dans un fichier dans un fichier compressé en `".tar.gz"`, un fichier README accompagne l'archive sous Linux pour expliquer où installer le plugin, ou comment rendre un fichier exécutable sous Linux

Pour MacOs c'est également une archive `".tar.gz"` qui est utilisée, ce format directement pris en charge sous Mac par StuffIt Expander, extrait directement l'application exploitable.

FIG. 18 – Installation du Plugin sous Windows.



8 Difficultés rencontrées

8.1 L'Async Reply

La programmation du graphisme sous linux (fenêtre, dessin) est gérée par la librairie Xlib. Or il se trouve que cette api n'est pas synchronisée et donc qu'un seul thread peut accéder à ces fonctions à la fois.

L'erreur "Unexpected Async Reply" est symptomatique de l'utilisation de la Xlib par plusieurs threads, et cette erreur stoppait aléatoirement le lancement du plugin dans netscape ; pourtant un seul thread semblait utiliser l'affichage. C'était en fait Netscape qui utilisait l'affichage sans prévenir.

De nombreuses solutions ont été envisagées pour contourner le problème. J'ai réalisé ainsi de nombreux tests comme initialiser l'affichage dans le même thread que celui fourni par Netscape lorsqu'il notifie une modification de la fenêtre (un objet Window), ou encore tenté de réaliser des temporisations dans le programme à certains moments critiques. L'idée la plus prometteuse a consisté à générer des événements quelconques pour qu'ils soient alors renvoyés par le système et y réaliser mes opérations d'affichage et de rendu. Là encore impasse car il semble que certaines fonctions ne doivent pas être réalisées pendant le traitement d'un événement.

La solution fut de passer par un XtAppContext généré par XtWidgetToApplica-

tionContext¹¹ et afin de profiter des méthodes XtAppLock, et XtAppUnlock qui permettent de prendre la main sur l’affichage (un peu comme un verrou pour les thread).

8.2 Des messages qui disparaissent

La gestion des événements pour le Plugin sous Linux dans l’exemple du GeckoSDK était réalisée par un Widget qui permettait à une méthode d’être appelée par un événement. C’est d’abord ce fonctionnement qui a été incriminé pour expliquer l’erreur de l’Unexpected Async Reply. En effet c’est un thread différent qui génère l’événement.

C’est pourquoi une méthode différente à été envisagée, elle consiste régulièrement à aller inspecter le nombre d’événements dans la file du système grâce à la méthode XPending et les récupérer avec la méthode XNextEvent. Cependant j’ai vite remarqué que des événements disparaissaient, ce qui n’est pas gênant pour ceux des déplacements de souris, mais est beaucoup plus ennuyeux lorsque c’est un relâché de bouton qui à été raté et que la caméra se déplace sans le vouloir.

La version stand alone qui utilisait la même méthode de gestion des événements, n’avait pas ce problème, et c’est donc Netscape (encore lui) qui scrutait la file d’événements et se les appropriait. Une fois l’Async Reply réglé, l’utilisation d’un Widget ne posa plus de problème et c’est la solution qui a été employée puisqu’elle délivrait bien tous les messages. Il est intéressant de noter qu’un Widget se situe au dessus de la Xlib et qu’il doit pourtant utiliser des fonctions standard pour arriver au même résultat.

8.3 La Glib pour VisualC++

La Glib était nécessaire pour la partie des sources réalisant l’affichage et la gestion de la structure de la scène. Ces sources existaient déjà et je n’ai pas eu à l’utiliser. Cette librairie permet d’utiliser des types standards qui auront la même taille pour tout les systèmes. Elle fournit également de nombreuses fonctions basiques comme l’allocation de mémoire ou des plus complexes, comme la gestion des listes chaînées.¹²

¹¹Un widget était déjà utilisé pour recevoir des évènements, voir section suivante.

¹²La Glib est particulièrement utilisée pour Gimp le logiciel de dessin open source

Il se trouve que les en-têtes de cette librairie ne se compilent pas avec VisualC++ qui n'emploie pas un langage C++ complètement standard, elle sont pourtant nécessaires pour l'édition des liens. VC++ génère en effet quelque centaine d'erreurs à la seule vue du fichier glib.h et la correction des premières erreurs m'a vite découragé.

La solution du chargement dynamique par l'ActiveX du plugin s'est révélée être la plus logique. Ainsi une seule version du plugin existe, qui se compile complètement avec MingW.

8.4 La Glib pour MacOs

Des problèmes sont également apparus avec l'utilisation de la Glib pour MacOs. Cependant, la difficulté était différente. Je n'ai pas trouvé dans un premier temps de versions compilées de cette librairie, nécessaire à l'édition des liens.

Il s'agissait alors de compiler directement les sources librement utilisables. Cependant l'environnement de développement de MacOSX n'est pas aussi souple que celui de Linux. Je me suis contraint à adapter les sources et y corriger les nombreuses erreurs ainsi que dans les options passées au compilateur, jusqu'à ce que ce que la librairie gtk vienne à manquer et qu'il faille à nouveau adapter les sources de cette librairie.

J'ai alors envisagé d'adapter les sources du projet pour en supprimer toutes références à la Glib, mais rapidement le travail s'est trouvé très important, il ne s'agissait pas simplement de transformer les `gint` en `int` ! Finalement j'ai pu trouver une version ancienne mais parfaitement fonctionnelle de la Glib qui régla tous les problèmes.

8.5 Passage de paramètre à l'ActiveX

Le passage de paramètre à l'ActiveX est une action assez complexe à mettre en œuvre. Malgré un grand nombre d'essais dans différentes directions, je n'ai pas su l'implémenter. Contrairement aux plugins Netscape, un ActiveX peut être utilisé en dehors d'une page internet, c'est donc un mécanisme de persistance qu'il faut implémenter. Une autre difficulté rencontrée vient du fait que lorsque des paramètres sont passés à un contrôle ActiveX, il est déclaré comme peu sûr par le système ; il existe cependant une méthode assez complexe permettant d'indiquer qu'un ActiveX est sûr.

9 Connaissances retirées

9.1 Organisation du travail

Dans le cadre du stage j'ai pu expérimenter une technique d'organisation du travail originale. Le principe est simple, il s'agit de prévoir la veille l'organisation de la journée du lendemain : décrire les tâches et prévoir le temps à y consacrer. Le lendemain on peut alors apprécier le retard pris dans son travail.

Cette méthode révèle la difficulté à prévoir la durée du travail, et je me retrouvais souvent avec quelques heures de retard. J'aurai appris que la programmation au niveaux des système d'exploitation est largement aléatoire : il n'est pas rare qu'une modification pour un système entraîne des bugs pour les autres.

Je reprocherais à cette méthode d'être trop ciblée sur la journée alors qu'un projet se prévoit généralement jusqu'à son terme. Cependant, j'ai pu constater qu'elle d'organisation permet d'améliorer son efficacité en matière de temps de développement. Elle oblige également à réfléchir aux différentes tâches du projet. Je recommanderais certainement cette méthode.

9.2 Les Makefiles

Les fichiers Makefile ont été particulièrement important tout au long du développement. Ces fichiers permettent la bonne compilation des sources jusqu'au fichier exécutable.

J'ai ainsi appris comment gérer les dépendances entre les différents fichiers objets utilisés pour l'édition des liens,¹³ comment appeler des méthodes du système pour, par exemple, lister un répertoire, appeler d'autre Makefile et inclure des définitions communes.

Ils m'ont permis d'explorer les différentes options passées au compilateur, la définition des en-têtes pour la compilation et celle des bibliothèques pour l'édition des liens.

¹³La gestion des dépendances permet de s'affranchir de tous make clean, cependant il faut parfois s'y résoudre.

9.3 Programmation des systèmes

La programmation pour plusieurs systèmes d'exploitation à été très instructive. J'ai appris en particulier toute les subtilités des systèmes de fenêtrage, le système MacOS qui fournit beaucoup de libertés mais qui laisse beaucoup trop de développement à la charge du programmeur, celui utilisé par Linux assez simple à comprendre mais qui nécessite une synchronisation des fonctions, et celui de Windows qui s'est révélé être le plus simple.

Le système de gestion d'événement est unique pour chaque système. La philosophie employée est donc assez différente, sous linux il est nécessaire de s'enregistrer auprès du système pour par exemple recevoir des informations sur les redimensionnements de la fenêtre ou encore la position de la souris.

J'ai utilisé plusieurs outils de développement comme VisualC++, MingW par gcc. La connaissance de ces environnements est bien évidemment un plus pour ma formation.

9.4 Programmation OpenGL

La programmation en OpenGL était quelque chose de totalement nouveau, j'ai largement profité des excellents tutoriaux du site NeHe (<http://nehe.gamedev.net>). Bien que le moteur de rendu m'était fourni, j'ai pu programmer des scènes rudimentaires proposant assez peu d'interaction pour me familiariser avec cette bibliothèque et tester la bonne mise en place de l'affichage OpenGL.

9.5 Programmation Réseaux

J'avais déjà une bonne expérience de la programmation des réseaux avec Java, cependant l'utilisation de fonctions sur les Socket en C était relativement nouvelle. J'ai appris particulièrement comment résoudre une adresse et obtenir l'adresse IP ; comment écrire des entiers d'une manière portable sur le réseaux grâce à la fonction `htonl` et son inverse `ntohl` notamment ; comment manier aussi bien des paquets TCP que UDP.

10 Remerciements

Je tiens à remercier particulièrement :

- Olivier Michel : pour l'encadrement du stage, ses enseignements techniques, et les précieux conseils sur mes travaux personnels.
- Auke Ijspeert : pour le suivi tout au long du stage et la confiance qu'il m'a accordé.
- Marlyse Taric : pour la préparation de mon arrivée à Lausanne.
- Alessandro Crespi : pour tous les connaissances qu'il m'a apportées dans de nombreux domaines.
- Jonas Buchli : pour tous ses conseils qui m'auront fait gagner beaucoup de temps.