Université de Marne-la-Vallée
Diplôme d'Ingénieur IMAC

Promotion 2005-2008

Christopher TARENTO

Rapport de stage
Juin - Juillet 2007

Société : Gameforge France
104 rue du Faubourg St Antoine
75012 Paris

Tuteur : Nicolas VIZERIE

Remerciements

Ce stage de deux mois représente, à mes yeux, deux enjeux majeurs. Tout d'abord, il constitue le premier de ma formation à l'IMAC mais également dans un cursus multimédia et informatique. Je tiens à remercier tout particulièrement Nicolas Vizerie et Typhaine Le Gallo de m'avoir fait confiance et de m'avoir offert l'opportunité de pouvoir découvrir le domaine du jeu vidéo à travers le projet Ryzom.

Ensuite, il était important de pouvoir réaliser ce stage dans les meilleures conditions possibles, et à ce titre, je voudrais remercier tout particulièrement Philippe Bauby et Daniel Miller pour m'avoir donner tous les éléments nécessaires au bon déroulement de celui-ci.

Je remercie également toute l'équipe de Gameforge France pour leur accueil et leurs conseils durant ces huit semaines.

Résumé

LORS de ce stage, ma mission a été de mettre au point un nouveau noeud dans le scene graph de la librairie graphique spécifique au jeu Ryzom : la NeL. Durant les huit semaines passées dans le studio Parisien de Gameforge France, j'ai pu me familiariser avec le développement d'un projet sur lequel j'ai travaillé en autonomie avec le soutien de mon tuteur Nicolas Vizerie. Les enjeux sont de taille et cette contribution n'est pas qu'un simple outil de production mais une véritable fonctionnalité qui pourra servir dans de futurs améliorations du moteur.

A insi, l'élaboration d'une nouvelle primitive de type texture projetée ainsi que la mise en oeuvre de son propre système de gestion évoluée auront occupé l'ensemble de mon immersion dans le milieu du jeu vidéo. J'ai pu découvrir et approfondir des domaines qui touchent à la performance graphique et à l'implémentation de solutions qui tirent partie de la puissance de l'accélération matérielle des cartes graphiques modernes.

Abstract

DURING this training period, my goal was to develop a new node in the Ryzom game graphic library : the NeL. During these eight weeks in the Parisian studio of Gameforge France, I became more friendly with code developping since I worked in autonomy thanks to my guardian Nicolas Vizerie. To my mind, this is a challenge and not a simple set of tools for production but a true improvement that can be use for revision of the 3D engine.

T hus, I spend all my training period in video game industry with the implementation of a new decal primitive and its own advanced management system. I could discover and learn a lot in many sides such as graphic performance tuning and highly accelerated solutions.

Table des matières

Chapitre 1

Introduction

1.1 Motivations personnelles

La formation d'ingénieur IMAC[1] nous propose un éventail d'enseignement très large dans des domaines d'activité qui vont du web à la production audiovisuelle, de la programmation orientée objet à l'histoire de l'art. Ainsi, durant ces deux dernières années, j'ai pu découvrir des matières qui m'étaient inconnues comme l'infographie et le développement d'applications de synthèse d'images et de réalité virtuelle.

Le jeu vidéo n'a pas encore été développé dans le cadre de la formation, c'est pourquoi j'ai cherché à réaliser ce premier stage dans un studio de jeu afin de découvrir et apprendre le maximum sur les coulisses d'un jeu. Certains aspects comme le fonctionnement et l'optimisation d'un graphe de scène ou encore le pipeline de rendu d'une carte graphique sont autant de généralités qui ne me sont donc pas inconnus. Je pourrai alors approfondir mes connaissances dans un cas pratique d'entreprise.

Actuellement, le jeu vidéo est un domaine qui occupe une place de premier ordre dans le monde du multimédia et l'effervescence qu'il dégage a une influence bien plus vaste. En effet, la recherche permanente sur l'amélioration visuelle, l'innovation et l'originalité pousse les éditeurs ainsi que les fondeurs à se mener une guerre sans merci pour proposer des solutions de plus en plus performantes et profiter de ces améliorations.

1.2 Mission : Decals rendering dans NeL et Ryzom

1.2.1 Besoins

- Actuellement, on ne dispose pas, dans la librairie NeL, d'objets instanciables permettant d'afficher une primitive de type *texture projetée*, souvent appelée *decal* ;
- La première application serait une matérialisation de la sélection d'un objet dans la scène pour une meilleure lisibilité de l'affichage ;
- Il éxiste de nombreuses applications dérivées pour agrémenter les effets spéciaux du jeu (effet d'auras, blood splats[2], traces de pas des personnages).

1.2.2 Éxistant

Ryzom Ring

Ryzom Ring est un outil d'édition de quêtes directement intégré dans Ryzom, il propose déjà un code d'affichage de textures projetées dans la scène, cependant :

- La réutilisabilité est faible (non intégré dans la librairie NeL, code spécifique au client) ;
- Le code ne bénéficie pas des propriétés développées dans le *scene graph* de NL3D[3] ;

[1] Image Multimédia Audiovisuel et Communication
[2] tâche projetée de sang lors de combats
[3] NeL3D, librairie graphique dédiée à l'affichage 3D

– L'implémentation ne respecte pas la cohérence du projet par la redondance du code et son intégration dans le client.

Le système de particules de NeL

Il propose le rendu de différents types de primitives (billboards, mesh, rubans), généralement en grand nombre. Il n'intégre pas non plus de primitives de type decal.

1.2.3 Objectifs

– Créer un nouveau noeud dans la librairie NL3D de type *texture projetée*. Comme les autres noeuds, celui-ci intégrera un support pour le clipping de caméra, l'animation de ses propriétés, le rendu ;
– Support basique : orientation et échelle ajustable, application d'une texture dans son intégralité ou en partie, choix d'une couleur, différents modes de répétition (clamp et repeat), différents modes de clipping de rendu (texture de masque et géométrique) ;
– La projection a une porté limitée sur l'axe des Z. Une zone de fondu est introduite près des bornes de l'intervalle ;
– Permettre un support réglable de l'alpha en fonction de l'orientation des faces projetées ;
– Ajout de différents modes de blending[4] en plus du traditionnel *alpha-blend* (modulate, add, add smooth) ;
– Facilité l'intégration au système de particules : un objet standard du même ordre qu'un mesh ou un billboard.

1.2.4 Contraintes

Matérielles

Le matériel visé est celui de la version de base de Ryzom, soit des PC équipés de GPU[5] de type GeForce2[6] en pipeline fixe avec deux étages de textures. Ces spécifications devenant vieillissantes, de nouvelles sont en cours de validation pour les évolutions de la librairie comme le support des shaders de deuxième génération (DirectX9).

Dans le doute, on décide de proposer deux versions du code : avec et sans vertex shaders[7] programmables.

Logicielles

– Fournir une solution qui favorise le batching[8] et le tri par texture afin de permettre l'affichage d'un grand nombre de decals ;
– Voir s'il est possible de modifier la version développée pour Ryzom ring ;

[4] caractéristiques et propriétés du mélange entre différentes faces transparentes ou semi-transparentes

[5] Graphic Processor Unit, processeur graphique

[6] génération de cartes Nvidia sortie en 2001

[7] Le vertex shader gère toutes les opérations à effectuer sur les sommets du maillage, soit les transformations géométriques et les calculs d'éclairement

[8] l'assemblage des decals dans une structure qui permet de tous les rendre en une seule passe

- Explorer la librairie NeL pour réutiliser et factoriser le maximum de code ;
- Une forme de projection de textures éxiste déjà : il s'agit des ombres des personnages. Elles fournissent un support pour déterminer les faces projetées sur le terrain et les objets. Toutes ces méthodes devront être utiliseés.

Chapitre 2

L’activité de Gameforge France

2.1 L'entreprise

, est la société créatrice et éditrice du MMORPG[1] Ryzom. Fondée en 1999 par David Cohen et Olivier Lejade, la société Nevrax a comme ambition de proposer une expérience de jeu différente et originale en produisant un MMORPG de type *science-fantasy*.

A son apogée, la société employait une soixantaine de personnes, et ce, dans tous les corps de métier nécessaires à l'élaboration complète d'un jeu : développeurs C++, développeurs Lua, game designers, level designers, web designers, testeurs, support client, administrateurs système, traducteurs.

Depuis son rachat en décembre 2006 par le spécialiste du jeu PHP Gameforge GmbH[2], Nevrax est devenu Gameforge France et a subit une restructuration. Ainsi, c'est une vingtaine de salariés qui continuent l'aventure et s'occupent de la maintenance du code et de l'ajout de nouvelles fonctionnalités.

2.2 La Nevrax Library, NeL

est une librairie de développement pour la création de jeu vidéo et plus spécifiquement de type MMORPG. En effet, elle est la base technique du projet Ryzom puisque qu'elle est développée et utilisée directement par l'équipe de Nevrax. Elle intègre donc une vaste arborescence, intégrant tout ce qui est nécessaire à l'élaboration d'un jeu, soit :

- moteur 3D (physique, éclairage, ombre, texture, shader HLSL, scene graph, animation, skinning, terrain, végétation) ;
- outils de production, éditeur de matériaux, plugin d'export 3DSMax ;
- moteur IA[3] pour l'évolution des PNJ[4] et les algorithmes de recherche de chemins ;
- moteur son 2D et 3D ;
- gestion réseau client/serveur ;
- mathématiques géométriques ;
- traitement d'images ;
- gestion des évènements et scripts Lua.

Elle est diffusée sous license GNU General Public License[5] et est téléchargeable directement via le CVS du site http ://nevrax.org. Ce choix de license d'utilisation et d'ouverture au monde libre a particulièrement permis de faire parler de la société et de lui voir attribuer une prime de la part du ministère de

[1] Massively Multiplayer Online Role Player Game, jeu de rôle en ligne massivement multijoueur

[2] OGame est le jeu de stratégie spatiale phare de la société

[3] intelligence artificielle

[4] personnage non joueur

[5] détails sur http ://creativecommons.org/licenses/GPL/2.0/

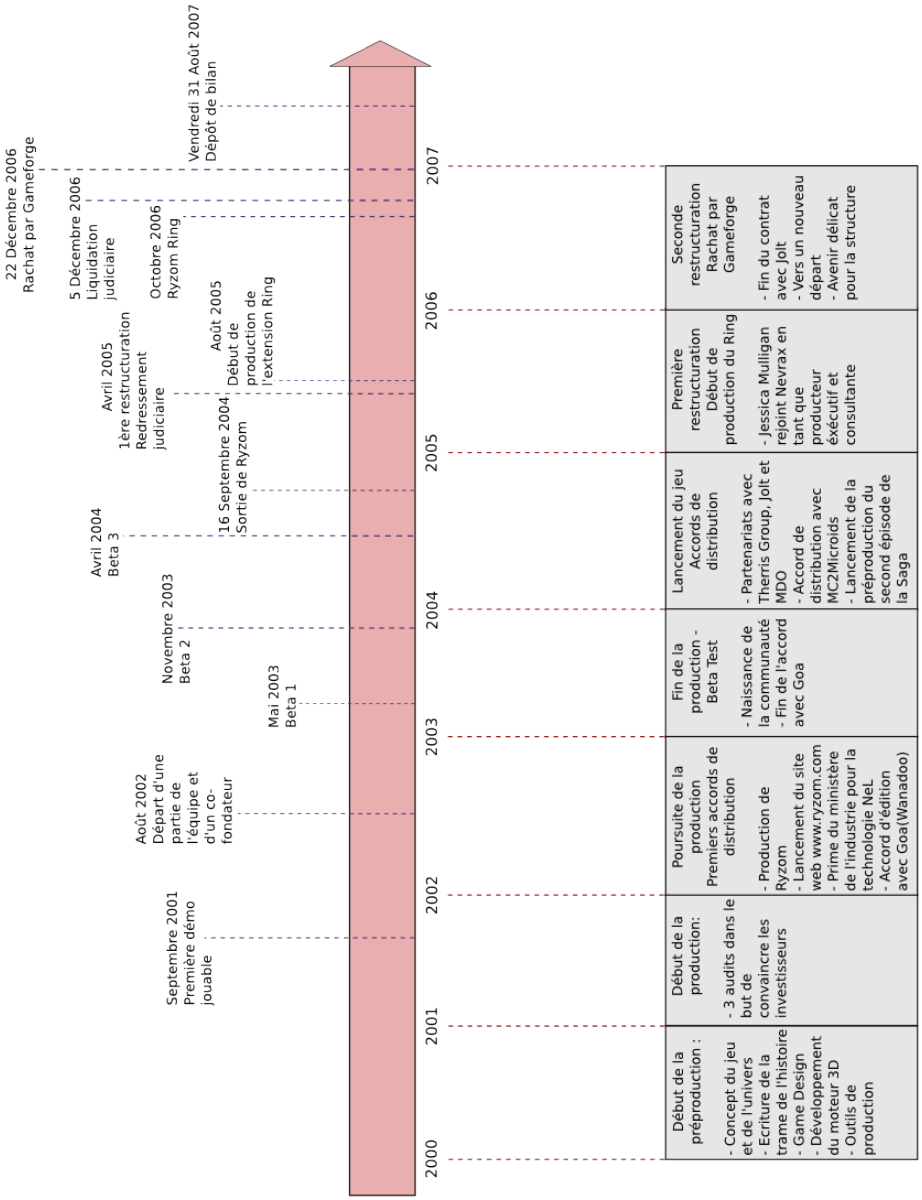


FIG. 2.1 – Historique

FIG. 2.2 – SnowBalls, jeu de démonstration utilisant la NeL

FIG. 2.3 – Visite de Richard Stallman au studio Parisien

l'industrie et l'attrait de la Free Software Foundation représentée par la visite de Richard Stallman en novembre 2001 (voir figure 2.3).

Bien que son développement et son utilisation principale se fassent sous un environnement Windows, la NeL a été conçue pour être multiplateforme. Néanmoins, de récents choix techniques notamment au niveau des shaders restreignent les dernières évolutions aux plateformes DirectX, excluant ainsi une utilisation Linux ou Mac.

2.3 L'industrie du MMORPG

2.3.1 Définition

Le domaine du MMORPG est un marché en pleine expansion depuis ces dix dernières années. Une des particularités du MMORPG est sa relative robustesse vis-à-vis du piratage. En effet, il est évident que nombre de jeux vidéo souffrent actuellement des copies illégales mais les jeux en ligne comptent sur un autre moyen de revenu : les abonnements. Ainsi, la plupart

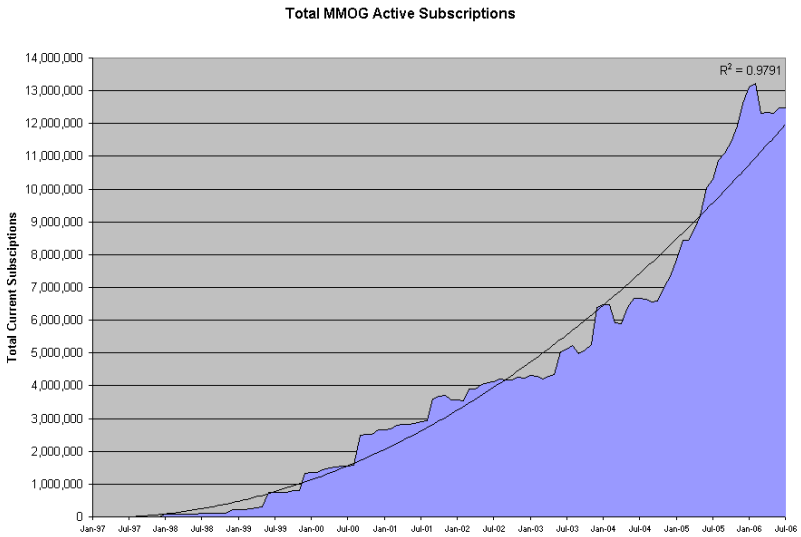


FIG. 2.4 – Nombre d'inscriptions mondiales aux MMORPG

des éditeurs propose d'essayer le jeu pour une période limitée à la fois dans le temps et dans l'espace, ne laissant entrevoir qu'une infime portion du monde virtuel.

De plus, il apparaît que les joueurs affectionnent particulièrement ce type de jeu car il permet de dépasser les frontières non seulement du point de vue physique puisque qu'on évolue dans un monde virtuel persistant[6] mais aussi culturel avec l'appartenance à une nouvelle communauté et surtout dynamique puisque le contenu même du jeu est destiné à être modifié et embelli par les joueurs. Nous sommes donc à une période où les enjeux et l'intérêt sont partagés par les éditeurs et les joueurs.

Enfin, l'industrie du MMORPG profite d'une certaine inertie où les temps de développement très longs ne permettent pas de sortir un titre tous les six mois. Il en résulte que les jeux ont une durée de vie relativement longue comparée aux jeux de FPS[7]. Si le jeu tient ses promesses en terme de stabilité et d'originalité, il ne fait aucun doute qu'il saura conserver une communauté de joueurs sur une période très longue comme on peut le voir avec *Dark Age of Camelot*, sorti en 2002 et dont le nombre d'abonnés frise encore les 30000.

2.3.2 Concurrence

L'intérêt financier est donc une réalité et le marché du MMORPG est un marché relativement jeune et prometteur où un projet de qualité peut trouver sa place assez facilement grâce à la portée internationale du jeu. Néanmoins, on ne peut que remarquer sur la figure 2.5 qu'environ 80% des parts de marché sont détenues par les cinq jeux les plus populaires.

[6] la partie dure indéfiniement, au-delà de son temps de connexion et l'environnement n'a pas besoin de joueur pour éxister et continuer à évoluer

[7] First Person Shooter, jeu de tir à la première personne

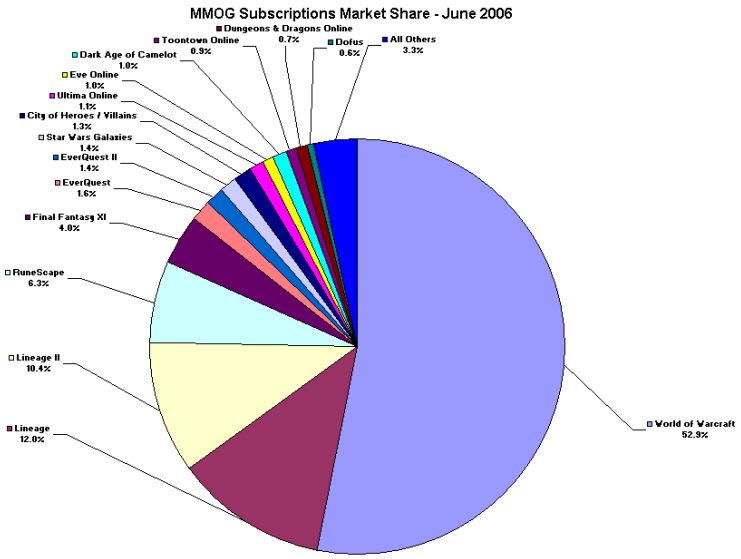


FIG. 2.5 – Parts de marché des différents MMORPG

2.4 Ryzom

2.4.1 Saga of Ryzom

L'univers incarné par Ryzom est un environnement fantastique évoluant sur une planète végétale et vivante, appelée *Atys*. Comme on peut le voir dans l'historique 2.1, le projet de Nevrax était de faire une Saga[8] de cet univers. Plusieurs épisodes se seraient succédés pour agrémenter et augmenter l'étendu du monde. Néanmoins, le projet sera renommé simplement *Ryzom* pour des raisons marketing et réajuster les prétentions du jeu après le départ du cofondateur David Cohen.

C'est ce goût pour le roleplay[9] qui pousse les joueurs à s'intéresser à Ryzom mais le manque de contenu dans les premiers mois et la sortie sur le marché du géant en devenir World of Warcraft[10] auront le pas sur l'innovation technique et visuelle que représente le jeu.

2.4.2 Le Ring

Le Ring est une extension que Nevrax a apporté à la communauté du jeu. Il s'agit d'un éditeur WYSIWYG[11] de quêtes et de missions pour Ryzom. De ce fait, la société devient la première au niveau mondial à proposer ce genre de

[8] « On appelle saga un récit en prose, toujours en prose, ce point est capital, rapportant la vie et les faits et gestes d'un personnage, digne de mémoire pour diverses raisons, depuis sa naissance jusqu'à sa mort, en n'omettant ni ses ancêtres ni ses descendants s'ils ont quelque importance », [Boy02]

[9] incarnation d'un personnage et évolution des actions en fonction des valeurs que réprésnte ce personnage

[10] Société Wizard

[11] What You See Is What You Get, ce que tu vois c'est ce que tu as

FIG. 2.6 – Éditeur de personnages du Ring

fonctionnalité pour un MMORPG. L'intérêt est de faire participer activement la communauté à la création et à la personnalisation de contenu.

Comme pour tout éditeur, il convient de pouvoir sélectionner des entités de décors, personnages afin de les déplacer, modifier, personnaliser. C'est ainsi qu'un effet visuel assez intéressant fait son apparition afin de mettre en évidence la sélection de tel ou tel autre objet : il s'agit des cercles de sélection projetés sur le sol.

Chapitre 3

Environnement de travail

3.1 Collaboration au sein du département *3D*

Mon stage s'est effectué au sein du département 3D mené par mon tuteur Nicolas Vizerie. Stagiaire chez Nevrax en 2001 alors qu'il préparait son diplôme d'ingénieur en informatique à l'ISIMA[1], il est aujourd'hui le responsable du département. Suite à la restructuration, ce service ne comporte plus que deux salariés, Nicolas et Typhaine Le Gallo ainsi que deux stagiaires, Florent Jaulin (école Creajeux) et moi-même. L'équipe est donc assez réduite, permettant de faire connaissance assez vite et de s'insérer facilement. De plus chaque département est centré dans une zone du studio, ce qui favorise le contact afin, par exemple, de poser des questions et d'ouvrir le dialogue sur des problèmes de développement sans déranger les autres collaborateurs.

3.2 Station de travail

A mon arrivée, un poste avait été préparé par l'administrateur système Philippe Bauby avec tous les logiciels nécessaires aux huit semaines de stage. Ainsi j'ai pu travailler sur une machine équipée de 2 Go de RAM ainsi que d'une carte graphique Nvidia 6800 GT. Du point de vue logiciel, l'environnement de développement est Visual Studio 6 agrémenté de STLPort[2] ainsi que du système de compilation IncrediBuild (voir section 3.3.2).

3.3 Compilation

3.3.1 Modes de compilation

Au moment de compiler la librarie NeL, il faut faire le choix du mode de compilation. En effet, différentes possibilités s'offrent à nous selon qu'on veuille profiter d'un affichage verbeux lors du débogage ou d'une application plus performante au moment de la validation de la fonctionnalité. En fait, une gestion avancée des macros sert à contrôler l'affichage de messages dans la console à l'aide de fonctions C au nombre de paramètres variable à l'instar de `printf(const char *, ...)`.

```
fonction nlwarning(...) :
  si DEBUG ou DEBUG_FAST
    afficher(...)
  sinon
    ne rien faire


fonction nlassert(condition) :
  si DEBUG ou DEBUG_FAST
    assertion(condition)
  sinon
    afficher(condition)
```

[1] Institut Supérieur d'Informatique de Modélisation et de leurs Applications
[2] la STL de visual 6 étant connue pour ses bogues, STLPort devient indispensable

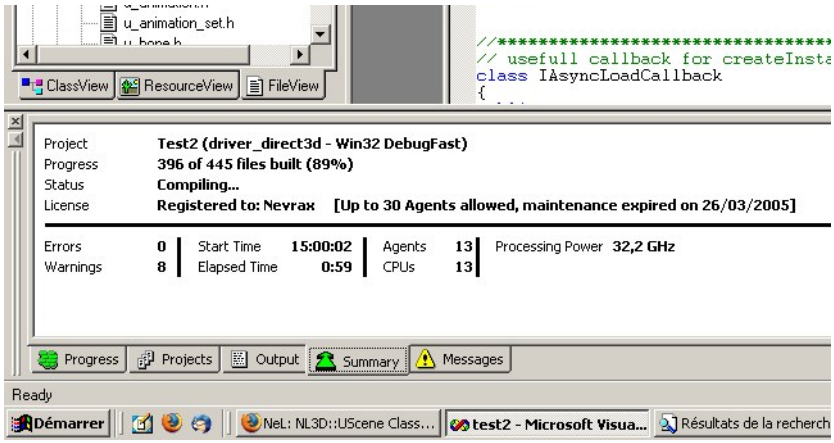


FIG. 3.1 – La compilation de la librairie avec 13 ordinateurs

Finalement, nous avons, par ordre de vitesse, la collection suivante :
- *Debug*, affichage verbeux et toutes les libraries en mode *Debug*
- *Debug Fast*, affichage verbeux
- *Release Debug*, code optimisé et affichage verbeux
- *Release*, code optimisé

La plupart du développement s'effectue en mode *Debug Fast* afin d'avoir un contrôle sur ce qui ce passe avec des performances honorables. En effet, le mode *Debug* n'est que rarement utilisé car la STL souffre alors d'opérations supplémentaires très lourdes rendant l'application très lente et un affichage inutilisable.

3.3.2 IncrediBuild

IncrediBuild est un gestionnaire de compilation développé par la société Xoreax. Il permet d'effectuer une compilation distribuée sur l'ensemble des machines de développement du studio, soit une douzaine de postes. Tous les fichiers sources qui peuvent être compilés de manière parallèle transitent par le réseau et les objets sont rapatriés sur le poste d'origine pour l'édition de liens et la création de l'éxécutable. Le temps de compilation passe donc de 15 minutes à 1 minute et 20 secondes (voir figure 3.1) pour compiler les quelques 440 fichiers[3] de la librairie NeL. Tout est totalement transparent pour l'utilisateur qui lance le processus de la même façon que pour une compilation standard.

3.4 CVS

3.4.1 Présentation

CVS est l'acronyme de *Concurrent Version System*. C'est le logiciel de gestion de version utilisé chez Gameforge. Il n'est pas question d'expliquer ici le

[3]NL3D et NLMISC seulement

fonctionnement de CVS mais plutôt de mettre en évidence l'organisation du projet autour de CVS.

Tout d'abord, l'ensemble du code source de la NeL ainsi que du projet Rysom est versionné avec l'intégralité du contenu comme les textures, les mesh ou encore les outils de développement. Néanmoins, un système de droits et privilèges régit l'accès aux différentes parties du gestionnaire. Ainsi, les collaborateurs de Nevrax ont un accès au contenu de Ryzom alors que les anonymes se connectant depuis l'internet ne peuvent télécharger que la partie open-source, soit la NeL. Tout le monde peut donc récupérer la branche de développement de son choix sur nevrax.org. La connexion au serveur *CVS* se fait par la simple commande suivante :

```
$ export 'CVSROOT=:pserver:login@cvs.nevrax.org:/home/cvsroot'
$ cvs login
$ cvs -z3 checkout code
```

3.4.2 Cycle de développement

La particularité de Gameforge est la manière dont chacun utilise le gestionnaire de version pour le développement du projet.

Pour commencer, il faut savoir que chaque salarié qui a accès au dépôt possède deux branches qui lui sont propres et qui constituent un cycle de développement pour une fonctionnalité. On distingue les branches :

- `WRK_*`, le travail en cours de développement jusqu'à validation des objectifs de la part du responsable de département ;
- `DONE_*`, le travail en cours de test et de validation de la stabilité et des performances de la part du département de *testing*.

Le préfixe de chaque branche est constitué des initiales du salarié ou de *ITx* pour chaque intérimaire/stagiaire (voir annexe .3).

La branche `HEAD` est la branche stable courante du projet. Les nouvelles fonctionnalités remontent dans cette branche après validation de la branche `DONE_*`.

Afin d'éviter des conflits entre versions, les opérations effectuées sont uniquement des *merge* qui remontent ou redescendent les branches comme on peut le constater sur le schéma suivant.

```
   /---------/---[dev]------[valid]-\----------------- WRK_*
  /          ^                      v
 /----/----/--------------------------\--[test]--\---- DONE_*
/    ^                                            v
----/-------------------------------------------------\-- HEAD
```

L'intérêt est de pouvoir fractionner le dépôt CVS de façon très ordonnée donnant ainsi un historique complet du travail de chacun et facilitant l'utilisation fréquente à des fins de sauvegarde des travaux en cours. Le responsable a ainsi une vision précise du travail de chacun des collaborateurs.

De plus un script post-merge envoie systématiquement le contenu de chaque *commit* à un développeur choisi aléatoirement afin qu'il s'assure que

l'opération effectuée n'est pas une erreur ou ne porte pas atteinte à l'intégrité générale du projet.

Chapitre 4

Textures projetées ou *decals*

4.1 Introduction

De nombreux moteurs concurrents proposent des textures projetées aussi bien comme simple primitive que comme éléments du système de particules. Si l'intérêt et la mise en oeuvre dans un jeu de type FPS semblent plus naturels, l'intégration dans un MMORPG comme *Ryzom* se fait de plus en plus ressentir aussi bien au sein du jeu comme un effet visuel que dans un éditeur comme *Ryzom Ring* pour faciliter la manipulation des objets et l'édition des quêtes.

Actuellement, le code permettant de mettre en oeuvre des textures projetées ne sert qu'au *Ring* car l'intégration est très brutale comme on a pu le contaster à la section 1.2.2. Le résultat est sans appel puisque les performances se trouvent excessivement dégradées à partir d'une quarantaine de decals, on passe de 85 fps[1] à 15 fps pour 200 decals. À première vue, cette dégradation spectaculaire est justifiable par un calcul à chaque image :

- la position de chaque decal ;
- la sélection des faces pour chacun des objets de la scène par les classes de gestion de collisions ;
- les coordonnées UV de chacun des sommets du decal ;
- un appel à la carte graphique pour chacun des decals.

Comme pour toute application graphique, deux facteurs limitants nous paraissent directement liés aux performances : la puissance de calcul du CPU et celle du GPU. Il convient donc de mettre en évidence dans quelle proportion chacun d'eux intervient dans cette chute de framerate[2] et de trouver des solutions de mise en oeuvre de la fonctionnalité ne souffrant pas de ces limitations.

4.2 Différentes approches

Pour commencer, il conviendra de définir la forme de base d'un decal, à savoir un carré. Celui-ci trouvera sa place sur le dessus d'une autre surface, en parfaite coplanarité. Néanmoins, il est évident que très peu de supports pourront être assimilés à des surfaces planes, ce qui obligera à sélectionner les faces du support concernées par le decal et d'en faire une copie, copie qui deviendra le mesh du decal. De plus, la superposition de deux surfaces parfaitement coplanaires est une opération dont le rendu est assez instable car le test de visibilité basé sur la profondeur n'est pas en mesure de donner un résultat strictement stable, oscillant entre -0.0 et $+0.0$ et faisant sans cesse changer l'état de visibilité à la fois sur une même surface et entre différentes images comme on peut le voir sur les figures 4.1 et 4.2.

Une étude préliminaire a permis de dégager deux grandes orientations pour résoudre ce problème. En effet, il apparait que la mise en place des decals peut être faite soit par passes successives en isolant la surface occupée par le décal sur le support ; soit par superposition du décal sur son support.

[1] frames per second, images par secondes

[2] taux de rafraichissement exprimé en images/seconde ou fps

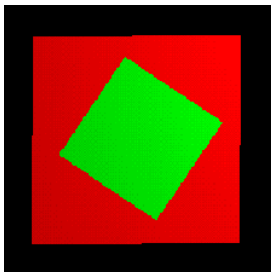


FIG. 4.1 – Superposition parfaite

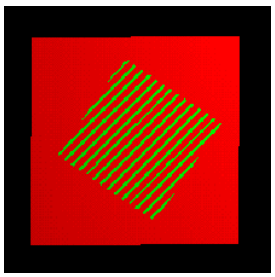


FIG. 4.2 – Apparition d'un artefact : le z-fight

4.2.1 Stencil buffer

Cette première méthode donne une réponse au problème en utilisant une option du pipeline de rendu : le stencil buffer. Il s'agit d'un buffer supplémentaire à l'instar du buffer de profondeur où l'utilisateur peut stocker des données propres à chaque pixel et selon un comportement qu'il peut définir. Ce bufffer rentrera ensuite en compte au moment des tests de visibilité (voir annexe .4). Un bon exemple d'implémentation a été trouvé chez SGI (voir [McR]) et pose le principe de base du système :

1. Activer le stencil ;
2. Paramétrer la fonction de test pour TOUT laisser passer ;
3. Paramétrer l'opérateur pour mettre 1 si le test de profondeur passe, 0 sinon ;
4. Dessiner le support ;
5. Paramétrer la fonction de test pour laisser passer si 1 ;
6. Désactiver l'écriture dans le stencil buffer ;
7. Désactiver le test de profondeur ;
8. Dessiner le decal.

Cependant, cette solution aussi intéressante soit-elle ne sera pas retenue, et ce, pour plusieurs raisons :

Compatibilité : le stencil buffer est un buffer auxiliaire et les cartes les plus anciennes ne le supportent pas complètement rendant son utilisation comme fondement assez risquée ;

Performances : l'utilisation d'un buffer supplémentaire et l'éxécution d'une série de tests additionnels ont une influence sur les performances de l'affichage ;

Implémentation : le stencil buffer est déjà utilisé pour distinguer les pixels qui représentent le positionnement du terrain.

4.2.2 Z-Bias

Le Z-bias est une fonctionnalité directement implémentée par les contructeurs de cartes graphiques. Il s'agit de modifier l'échelle de profondeur de manière ponctuelle pour *tricher* sur la position réelle d'un pixel et modifier le comportement lors des prochains tests de visibilité basés sur la profondeur. Cette méthode, expliquée dans [DeL01], est celle qui sera retenue pour le développement des decals.

Bien sur, il existe d'autres méthodes comme le suggère [MO] mais le Z-Bias est très simple de mise en oeuvre et confère une grande souplesse d'utilisation. De plus, l'environnement graphique de la NeL propose le rendu d'ombres dynamiques sur ce modèle de Z-Bias, ce qui facilite l'intégration des decals selon la même philosophie de mise en oeuvre de la librairie, tout en profitant de fonctions éprouvées par la pratique.

4.3 Performance

4.3.1 Bande passante

L'appel aux fonctions de dessin comme `glVertex3f()` est quelque chose d'assez ambigu sur le plan des performances. En effet, même si les temps d'éxécution sont extrêmement faibles, le système montre vite ses limites lors d'un nombre d'appels important. En fait, nous sommes plus confrontés à des problèmes d'interruptions systèmes que de manque de ressources graphiques. Lors de chaque appel à ce type de fonction, l'application dialogue avec le driver graphique qui transfère des données à la carte graphique afin d'éxécuter l'affichage de ce qu'on a demandé. C'est à ce niveau qu'il y a un véritable goulot d'étranglement[3].

Le point sensible du travail à effectuer sera donc de maîtriser au mieux ce passage délicat afin de faire économiser le maximum de bande passante au système. Un développement sachant tirer parti de ce constat sera à même de donner le meilleur du matériel, et ce, quelque soit sa configuration et sa vétusté[4].

4.3.2 Fill Rate

Un autre point particulièrement important est la gestion du fill rate. Le fill rate représente le nombre de pixels que peut rendre une carte graphique en une seconde. Les cartes actuelles ont une capacité de l'ordre du gigapixel, ce

[3] souvent qualifié par son homomnyme anglais *bottleneck*

[4] il ne faut pas perdre de vue la configuration cible vue à la section 1.2.4

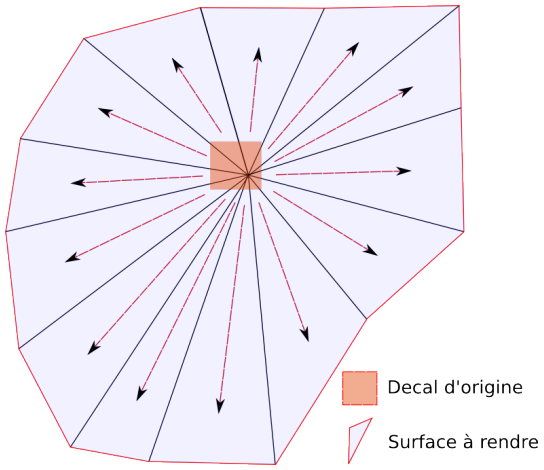


FIG. 4.3 – Cas le plus défavorable, on déborde sur des surfaces inutiles

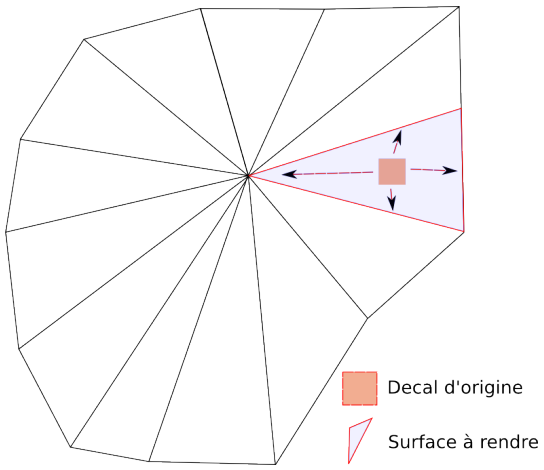


FIG. 4.4 – Cas moins défavorable, mais toujours du débordement

qui peut paraître impressionnant, mais si ces ressources sont mal utilisées, l'efficacité escomptée ne sera pas au rendez-vous.

Pour illustrer notre problème de fill rate, il convient de mettre en évidence les cas les plus défavorables comme on peut le voir dans les figures 4.3 et 4.4. On remarque que si le decal chevauche des triangles de grande surface, comme pour le terrain par exemple, le rendu d'un simple decal contraint la carte graphique à rendre ces surfaces inutiles et consommatrices de fill rate surtout si on pense à l'intégration au sein du moteur de particules.

4.4 Processus de rendu

4.4.1 Design-Pattern Model-Observer-Traversal

Toute scène dans la librairie NeL est organisée selon le design-pattern Model-Observer-Traversal, MOT. C'est un pattern de moteur 3d général. Ce pattern est intéressant car il permet une évolution facile d'un moteur grâce à une structuration séparant les données de leur traitement.

Model : un objet qui a une représentation dans le monde. Cela peut être un mesh, un effet...Un modèle en lui-même ne contient que des données. Il ne sait pas comment se dessiner.

Observer : ils sont organisés sous forme d'un graphe. Chaque observer référence un modèle, et un modèle est référencé par un nombre fixé d'observers (autant que de traversals). Un observer permet d'effectuer une tâche précise sur un modèle comme le rendu ou la détermination de visibilité ou l'animation. On a ainsi une séparation entre les données, et les algorithmes s'appliquant sur ces données.

Traversal : c'est une fonctionnalité du moteur. On pourrait l'appeler passe de rendu. Dans d'un traversal, tous les observers associés à un traversal sont exécutés lors d'une passe. A partir du graphe d'observers d'un tarversal, on obtient un graphe d'observers pour le traversal suivant. Ainsi, après le traversal de clipping par exemple, il ne reste que les objets visibles dans le graphe.

La NeL comporte un nombre impressionnant de ces traversals pour gérer différentes caractéristiques comme le clipping[5], l'éclairage, l'ombrage, l'animation, le niveau de détail ou encore le rendu comme on peut le voir à la figure 4.5.

Toute la puissance de la programmation orientée objet et en l'occurrence du *C++* va permettre la surcharge et la spécialisation du code pour notre classe `CDecal`. Il suffira de surcharger le clip traversal de la classe mère `CTransform`. En effet elle laisse passer tous les objets par défaut. Ainsi, le comportement de chacun des traversals peut être laissé neutre ou adapter à des besoins spécifiques comme ceux qui vont suivre.

4.4.2 Clip traversal

Ce traversal élimine les modèles qui ne sont pas visibles. Il se base sur le frustrum[6] ainsi que la distance de visibilité maximale (voir annexe .6.1).

L'utilisation d'une sphère englobante[7] est un choix de performance. En effet, ceci permet de réduire le nombre de tests et leur complexité pour déterminer si le decal est visible ou non.

4.4.3 Render Traversal

Celui-ci s'occupe du rendu à proprement parlé. Cependant, ce n'est pas parce qu'il est le dernier de la chaîne de rendu qu'il effectue lui-même le

[5] méthode consistant à ne calculer que le rendu des éléments visibles dans le cône de vision
[6] cône de visibilité
[7] appelée aussi bounding sphere

rendu des objets. En fait, le sytème mis en place pour les decals profite d'une dernière étape qui est le *manager* de decal, soit un *observer*.

C'est le manager qui aura comme tâche d'optimiser le traitement des decals en prenant en considération les points vus en section 4.3.

4.5 Implémentation

L'affichage d'un decal ne se fait pas aussi simplement que celui d'un mesh ordinaire. Il est caratérisé par une forme de base cubique qui représente le volume de projection, ou encore le volume dans lequel tout objet recevra sa projection. De cette manière, le decal est délimité par une boîte englobante[8] qui arrête le decal de façon exclusive, et ce, selon les trois axes.

Comme on peut le constater à la figure 4.5, différentes étapes sont donc nécessaires à la transformation d'une bounding box associée à une texture pour en faire une texture projetée opérationnelle.

4.5.1 Sélection de faces par Quad grid

Contrairement au quad tree, le quad grid est une simple grille dans laquelle on référencie des objets. Cette méthode très largement employée dans la NeL a comme particularité d'être très rapide et d'avoir la compléxité algorithmique la plus intéressante en $O(1)$. Néanmoins, le quad grid n'a ni de centre géométrique, ni de limite spatiale malgré un nombre de cases fixé. Il en résulte qu'il n'y a aucune garantie que la face sélectionnée soit réellement dans notre bounding box et non à quelques unités de là. Ceci s'explique simplement par le fait que l'insertion dans la grille se réalise par *modulo* pour avoir des objets référencés qui se situent dans la grille.

Cette structure convient donc à une sélection rapide des faces concernées par la projection mais avec une incertitude notable sur la position exacte des faces selon les trois axes. Il faut donc effectuer un raffinage à l'aide des plans de clipping que forment la bounding box du decal (voir annexe .6.2). Même si un test pour chacun des six plans doit être fait pour un nombre apparemment grand de faces inutiles, l'expérience montre qu'il est plus efficace d'utiliser une telle méthode.

4.5.2 Clipping

Les faces qui viennent d'être sélectionnées ne sont pas parfaites puisqu'elles sont une simple copie des faces qui sont touchées par la bounding box du decal et non la seule portion utile de la face comme nous l'avons vu en section 4.3.2. De plus, il ne faut pas oublier que notre but est de plaquer une texture sur ces faces et qu'il risquerait d'y avoir un débordement de texture sur la zone inutile comme on peut le voir sur la figure 4.6. Il y a donc une nécessité à clipper les faces pour deux raisons majeures aussi bien du point de vue esthétique que performance.

Deux méthodes auront été développées pour permettre d'adapter au mieux la sélection des faces. En effet, il faut trouver un juste équilibre pour

[8] appelée aussi bounding sphere

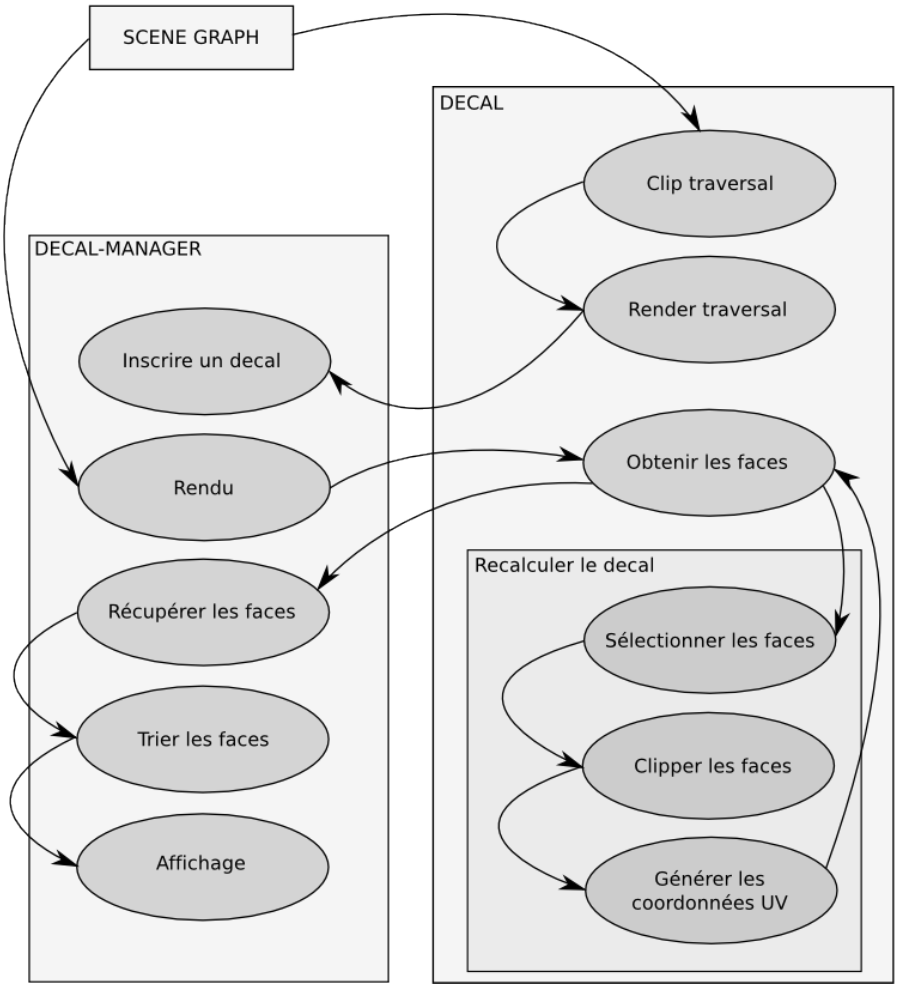


FIG. 4.5 – Traitement d'un decal

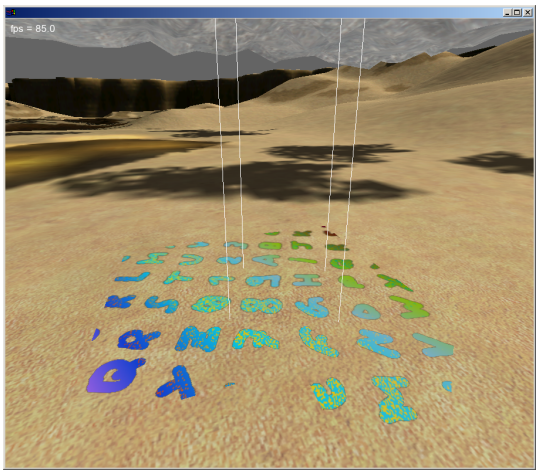


FIG. 4.6 – Pas de clipping, débordement de la texture

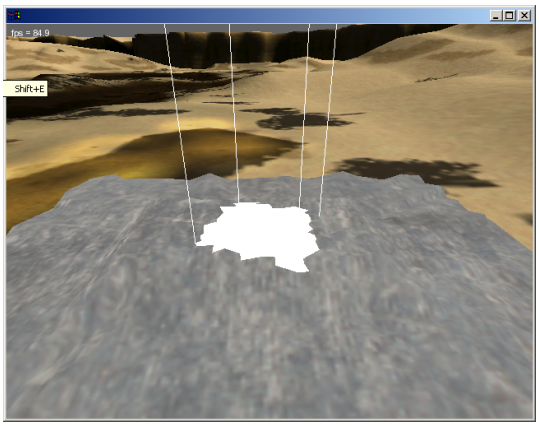


FIG. 4.7 – Sélection sur surface aléatoire

ce problème de clipping. D'un côté, il paraît intéressant de modifier la géométrie des faces récupérées pour les adapter parfaitement à la bounding box du decal afin de ménager au mieux le fillrate. D'un autre côté, cet accroissement de calculs risque de nuire aux ressources CPU dans le cas où le gain réalisé ne serait pas suffisant pour amortir sa réalisation.

Mask clipping

La librairie NeL autorise l'utilisation de deux étages de texture. Il pourraît être judicieux d'employer une d'entre elles comme masque afin de délimiter les bords du décal. De plus, une texture générée en interne de 4x4 pixels devrait être suffisante si on souhaite juste des bords nets et elle aurait un impact nul sur les performances. La géométrie des faces reste intacte comme on peut le constater sur les figures 4.7 et 4.8 mais le résultat obtenu à la figure 4.9 ne laisse rien entrevoir.

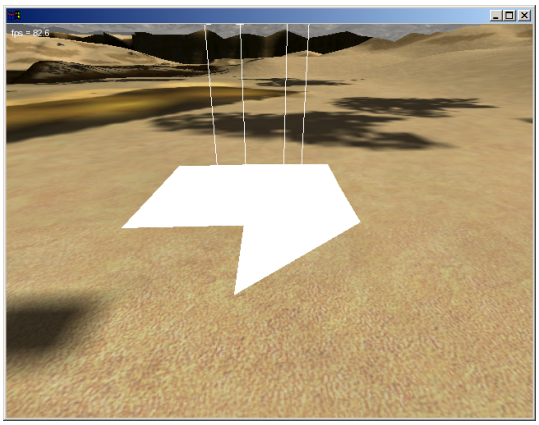


FIG. 4.8 – Sélection sur surface plane

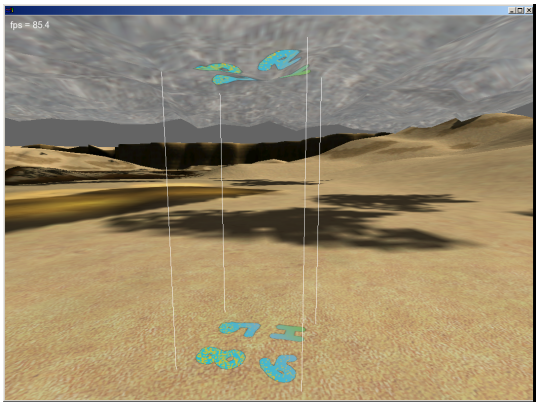


FIG. 4.9 – Résultat du mask clipping

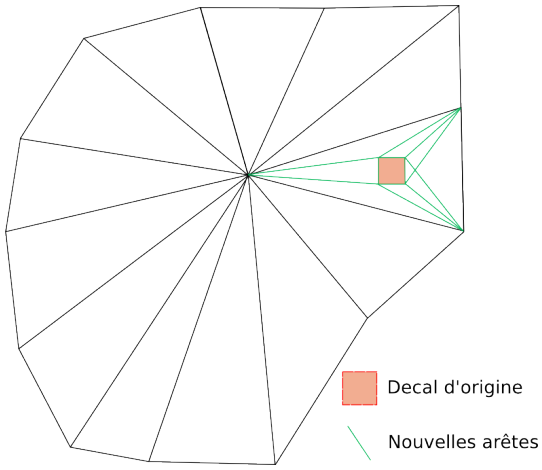


FIG. 4.10 – Modification du mesh, contruction de nouvelles arêtes

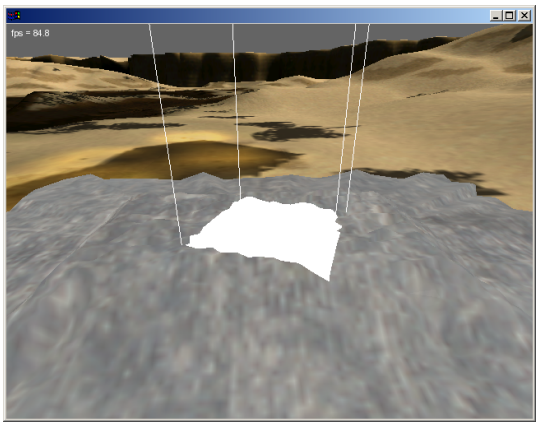


FIG. 4.11 – Vertex clipping sur un objet

Vertex clipping

La seconde approche consiste à modifier de façon géométrique les faces sélectionnées afin d'extraire uniquement la portion qui nous intéresse. Cette méthode est assez consommatrice pour le système car elle clippe chacun des sommets par chacun des plans de la bounding box. Le résultat intermédiaire obtenu en figures 4.11 et 4.12 montre bien que le decal ne déborde plus et qu'il est parfaitement délimité par la bounding box d'origine.

4.5.3 Génération des coordonnées UV

Les faces acquises précédemment ne renseignent que de position absolue ou relative[9] et il est alors nécessaire de trouver un moyen pour générer les coordonnées de textures correspondantes à chacun des sommets.

[9]les processus de recherche de faces et de traitement s'opèrent dans des repères différents selon qu'il s'agissent du terrain ou d'un objet, repère monde *World Matrix* ou local *Instance Matrix*

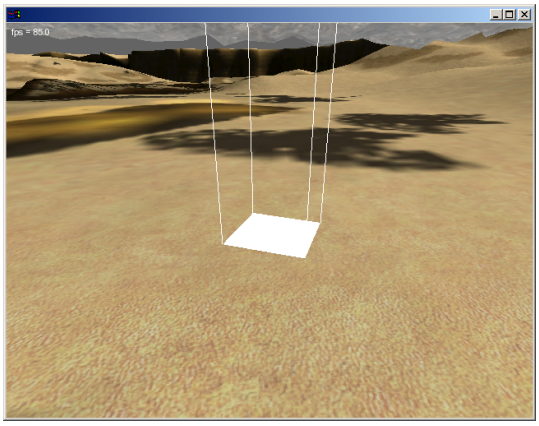


FIG. 4.12 – Vertex clipping sur le terrain

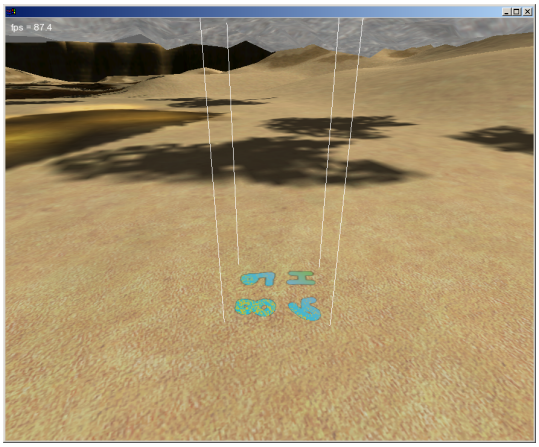


FIG. 4.13 – Résultat du vertex clipping

La bible de l'OpenGL [SWND06] donne quelques indices quant à la résolution de ce problème. Il existe un mode de génération automatique de coordonnées de texture basé sur les coordonnées, exprimées en local, des sommets. Cette idée est une piste remarquable par sa simplicité de mise en oeuvre car il suffit de connaître les coordonnées des sommets et la matrice de transformation permettant de retourner dans un repère local. C'est pourquoi, il a été choisi comme base de decal un cube de côté unitaire et dont les coordonnées des sommets ramenés en 2D[10] représentent directement leurs coordonnées UV. En tant que *model* à part entière du scene graph de la NeL, il est très facile de connaître la matrice de transformation cumulée de notre decal.

Néanmoins, le principe sera utilisé pour la génération de coordonnées de texture mais ce soin ne sera ni laissé à OpenGl, ni à Directx car il n'existe pas de texture unique pour l'ensemble des decals. En effet, si cette solution est appréciable pour des decals pris indépendamment, il devient impossible de l'utiliser comme on va le voir dans la section 4.5.4.

4.5.4 Batching

Comme on peut le remarquer sur la figure 4.5, le gestionnaire de decal[11] est au centre du processus de rendu. Le decal est donc bien un *model* contenant des données mais ne sachant pas les interpréter, soit dans notre cas en faire un affichage 3D. Lors du rendu d'une nouvelle image, le scene graph parcourt ses noeuds et appelle successivement le clip traversal puis s'il passe, le render clip. A cet instant, le decal ne réalise qu'une seule opération, l'inscription dans le decal manager. Quand le scene graph appellera l'affichage de ce dernier, il aura comme tâche de rapatrier l'intégralité des faces de chaque decal de façon ordonnée.

La création d'un decal ne peut se faire sans la création préalable d'un matériau. Chaque matériau enregistré dans le decal manager possède un identifiant unique, identifiant qui sera utilisé lors de la création de decal. De ce fait, les decals sont déjà naturellement classés par matériau, préparant le travail d'optimisation de l'affichage.

Heuristique de regénération des decals

Dans le cas de decals considérés comme *statiques*, il serait avantageux de modifier la géométrie des faces des decals de manière définitive. Ainsi, le fillrate serait optimal et l'utilisation CPU également. Dans un premier temps, il est assez aisé de mettre un flag[12] indiquant que c'est un decal statique. Dans ce cas, le calcul des faces du decal ne se fera qu'une seule fois.

Un decal *dynamique* ne pourra pas profiter de cette optimisation et il sera recalculé à chaque image. Dans ce cas précis, on pourrait même se demander s'il n'est pas plus intéressant d'utiliser un clipping par masque pour soulager le CPU. L'utilisation d'un compteur incrémental qui compte le nombre d'images durant lesquelles la matrice de transformation du decal est restée

[10] la coordonnée d'altitude est tout simplement ignorée
[11] appelé aussi *decal manager*
[12] drapeau. Une caractéristique ne prennant que deux valeurs, vrai ou faux.

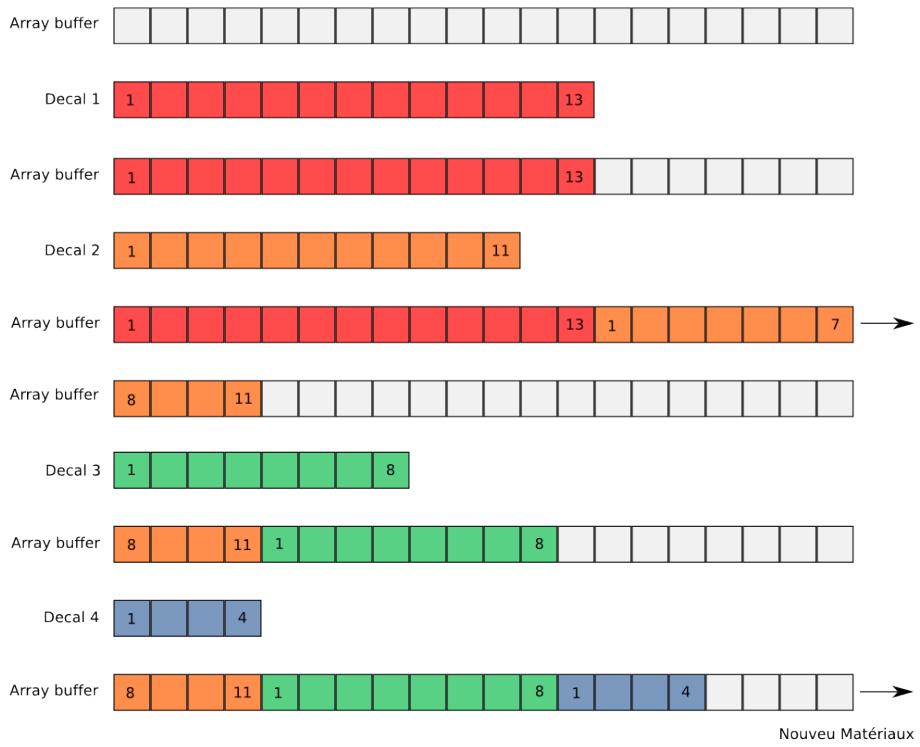


FIG. 4.14 – Gestion du *Array Buffer*

identique va permettre de repérer les decals qui auraient une tendance statique et les faire profiter de l'optimisation.

Array buffer

Afin de prendre en compte les remarques vues en section 4.3, le but du manager est de rendre l'intégralité des decals dans un minimum d'appels aux fonctions graphiques. Pour ce faire, l'affichage à proprement parlé des decals ne se fait que par l'intermédiaire d'un *array buffer*. Le [SWND06] apporte toutes les indications quant aux possibilités offertent par cette méthode de rendu mais un point très important sur le stockage en mémoire ne pourra être mis en évidence que par l'expérience de Nicolas Vizerie.

L'ensemble des coordonnées des sommets et de leurs coordonnées de texture associées est regroupé dans un tableau unique. Pour des raisons techniques, ce tableau a une taille fixée à l'avance. Les decals doivent donc être ajoutés judicieusement pour rendre à la fois le array buffer par matériau et sans dépasser sa capacité. En cas de dépassement du nombre de blocs[13] disponibles, il faut rendre le tableau intégralement rempli puis le remplir avec ce qu'il reste et le rendre à nouveau comme on peut le voir sur la figure 4.14.

Un autre facteur d'influence sur les performances et qui est malheureusement que peu documenté dans la littérature est la zone de mémoire utilisée

[13] un bloc contient les coordonnées du sommet (X, Y, Z) et les coordonnées de texture (U, V)

pour stocker le array buffer.

RAM : mémoire système. La carte graphique en a un accès assez lent mais l'écriture y est très rapide la rendant adéquate pour des données devant être souvent raffraichies.

AGP : zone de la mémoire système dédiée à l'échange avec la carte graphique. L'écriture y est aussi très rapide et l'accès par la carte graphique profite au mieux de la bande passante du bus AGP ou PCIExpress. C'est un bon compromis pour des données assez hétérogènes et dont l'utilisation n'est pas expressément connue.

VRAM : mémoire vidéo intégrée à la carte graphique. L'écriture y est lente et unique. On ne peut pas modifier les données. Il faut désallouer et réallouer un nouveau tableau. Elle est particulièrement adaptée pour des données statiques et très fréquemment utilisées par la carte graphique.

De plus la zone mémoire utilisée peut également profiter d'un mode *Volatile* qui signifie que le tableau est doublé à l'instar du double buffer d'affichage afin de basculer facilement de l'un à l'autre et ne pas attendre que le premier soit fini d'être traité pour commencer à remplir le second. Ainsi, le processus bloquant de rendu graphique ne risque plus de ralentir notre manager quand il brasse des centaines de decals.

4.6 Résultats

4.6.1 Instabilités de positionnement

Lors des premiers essais des decals, un phénomène ressemblant au *z-fight* a fait son apparition malgré l'utilisation de *Z-Bias*. Cette anomalie vient du fait que les matrices de transformation utilisées dans le scene graph peuvent tout à fait être à la fois très grandes ou très petites. Il n'est pas rare, en voyant l'ampleur du monde, d'obtenir des objets positionnés à une position de $(160500, 2900, 300)$. Accumulées, les différentes matrices deviennent grandes à tel point que le résultat d'une multiplication entre elles devient instable à cause du manque de précision des nombres flottants.

Lors du test de visibilité au sein du pipeline de rendu, un autre problème vient s'ajouter à cela, il s'agit de la précision du buffer de profondeur. En effet, s'il est dejà possible d'atteindre les limites de précision d'un flottant codé sur 24 bits, qu'en est-il sur les pixels du buffer de profondeur qui peuvent, selon l'implémentation du constructeur de carte, être eux codés sur 8, 16, 24 ou 32 bits si on considère la palette que doit supporter la NeL.

Cette accumulation d'imprécisions est donc à l'origine de notre défaut d'affichage. La solution employée pour le résoudre a été de soustraire la position de la caméra à notre matrice de decal afin de revenir dans des coordonneés raisonnables pour l'ensemble de nos traitements. Néanmoins, ceci nous force à recalculer tous les decals au même instant à cause de la façon dont le manager traite le *batching*. L'impact sur les performances reste cependant très limité car il n'est pas nécessaire de les recalculer à chaque déplacement de caméra. En fait, la distance maximale de visibilité est une bonne base pour revoir notre heuristique car si on retarde le recalcul jusqu'à cette distance, les decals concernés ne seront plus visibles et n'auront plus besoin d'un recalcul.

FIG. 4.15 – Texture de base

4.6.2 Mipmaps

Afin d'opimiser encore la gestion de la mémoire et les appels aux fonctions graphiques, il a été choisi de faire un regroupement de textures pour stocker plusieurs images dans une seule de taille plus importante. Ceci pourra permettre d'intégrer facilement de l'animation pour un decal en choisissant des portions de texture différentes correspondant à des positions clés différentes par exemple. Les figures 4.15 et 4.16 illustrent cette idée.

La génération automatique de *mipmaps*[14] intégrée dans la NeL risque de poser problème en débordant sur les portions voisines lorsque le facteur de réduction du mipmap atteint la proportion que représente une portion de texture. Il faut donc limiter la génération de mipmap à ce seuil où les résultats esthétiques dénoncent un défaut de développement.

4.6.3 Rendu et performance

Les tests réalisés ont montré que le nouveau système de decal permet d'afficher environ 1000 decals à 60fps en mode de compilation *Debug Fast* et 1200 decals en mode *Release*. On atteint les 10000 decals pour 15 fps. Il y a donc un facteur d'amélioration considérable, entre 20 et 30. Il devient donc envisageable d'utiliser la nouvelle primitive decal au sein du moteur de particules mais plus généralement dans le jeu *Ryzom*.

[14] méthode qui consiste à éviter les effets de pixellisation en adaptant la résolution de la texture à la distance nous séparant de l'objet

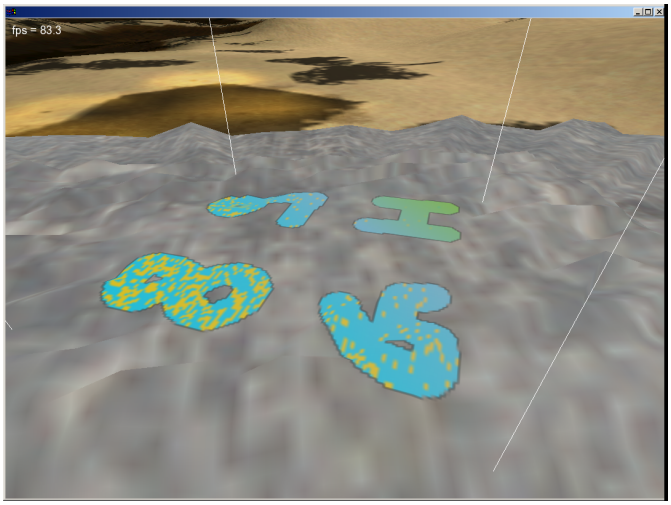


FIG. 4.16 – Seulement une portion de la texture est projetée

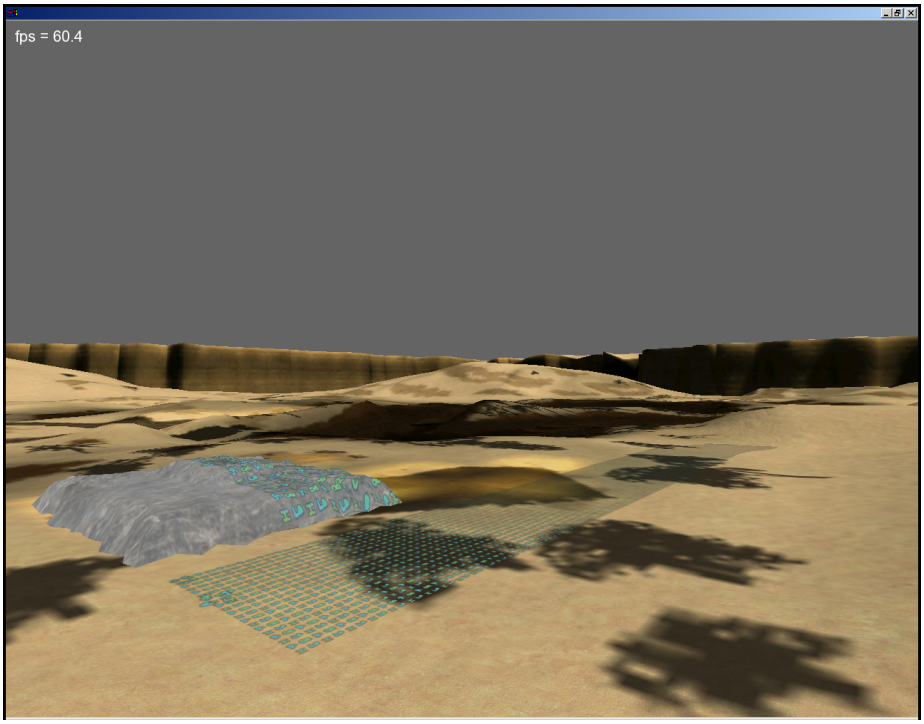


FIG. 4.17 – Un champ de 1000 decals

4.6.4 Améliorations possibles

Meilleure implémentation du manager

Le manager souffre encore de quelques défauts et principalement de quelques bugs sur le cas particulier de la première image. En effet, le premier passage dans le scene graph pose problème car les matrices renvoyées sont erronnées. La mise à jour des informations se fait toujours avec une image de retard, ce qui implique que les decals statiques aient été calculés avec une mauvaise position, ne les rendant visibles qu'après un parcours dans le monde pour actualiser les matrices et forcer un recalcul d'après l'heuristique de déplacement de la caméra.

L'heuristique de rafraîchissement souffre d'une certaine légèreté et il arrive que certains decals créés à la volée en cours d'utilisation ne prennent pas bien en considération la distance de visibilité.

Vertex Shader

Le temps restant n'a pas permis de s'intéresser à la programmation de vertex shader pour mettre en place des effets de fondu sur les bords du decal et plus particulièrement au niveau de l'altitude. De plus, la NeL subit actuellement une refonte complète de son système de shader *assembleur* pour une évolution vers le *HLSL*. Il aurait pu être intéressant d'étudier le passage de l'un à l'autre et particulièrement du point de vue de la génération de code *assembleur* à partir de code *HLSL*.

Chapitre 5

Bilan

Ces huit semaines de stage se sont déroulées très vite, à tel point que j'ai dû me contenter de ne développer que l'essentiel de la mission qu'on m'avait confiée. La mise au point d'un noeud de decal au sein de la librairie NeL fut pour moi une opportunité qui n'a d'égal que tout ce qu'on a pu m'apprendre aussi bien durant les phases de conception que de développement. Je ne sais pas si le travail que j'ai fourni parmi l'équipe de Gameforge France sera complété et utilisé jusqu'à finir dans la branche *HEAD* mais j'ai pu mettre en oeuvre aussi bien ce que j'ai appris durant ma formation à l'IMAC au niveau C++ et synthèse d'images que tous les conseils avisés que Nicolas a su me donner.

Ce stage représente malgré tout l'insertion dans un immense ensemble, celui de la librarie NeL. Des dizaines de miliers de codes, des centaines de classes et seulement une documentation et l'expérience de mes collaborateurs pour m'éclairer sur ce qu'on peut y faire et à quel endroit chercher. Ce projet de texture projetée est bien plus qu'un projet au milieu de tant d'autres pendant mon cursus d'ingénieur car il représente à la fois le premier stage dans le domaine du développement logiciel mais surtout un des rares projets où j'ai été mis en situation de complète autonomie. Ce fut donc un challenge très agréable à relever et je suis pleinement satisfait du sujet qu'on m'a proposé et de ce que j'ai pu en faire.

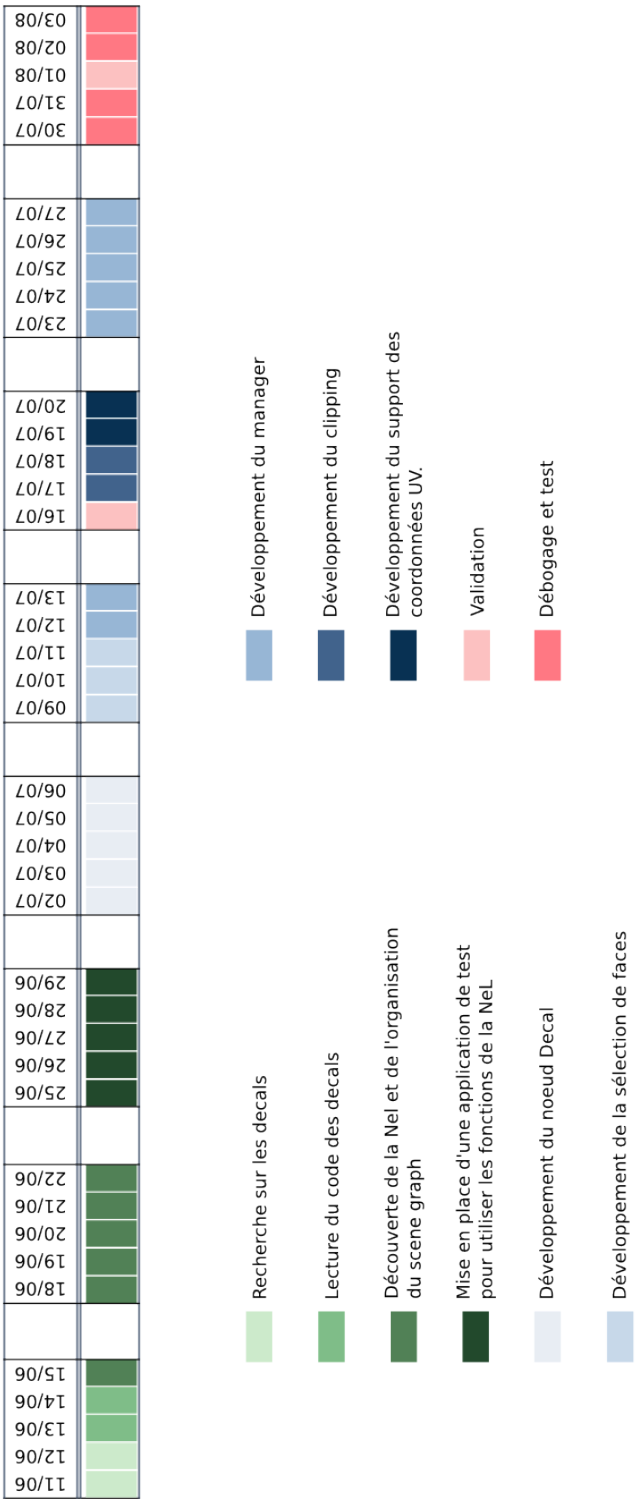
N'étant pas un joueur invétéré, ce fut une excellente occasion de découvrir un milieu en pleine effervescence où le principal mot d'ordre est *plaisir* du joueur. J'ai pu me rendre compte qu'il ne suffit pas d'être un excellent codeur, un excellent graphiste ou un excellent scripteur pour réussir dans le milieu mais aussi être débordant de culture de ce qui se fait dans le domaine. Le studio Parisien abrite une équipe de spécialistes du jeu, de son histoire, de son actualité mais aussi de sa technicité, ce qui fut sans doute pour moi la chose la plus déroutante.

À l'heure où j'écris ce présent rapport, je ne peux que déplorer le fait que le studio soit de nouveau en liquidation judiciaire. Mes pensées vont à l'équipe qui doit une nouvelle fois traverser une épreuve difficile qui met en péril le développement d'une part d'une librairie éprouvée et performante et d'autre part d'une autre vision du jeu en ligne. Il ne fait aucun doute que la structure devrait être à nouveau reprise car on ne peut que remarquer la motivation et la compétence de l'équipe qui constitue un des rares MMORPG Français.

Chapitre 6

Annexes

.1 Planning


11/06
12/06
13/06
14/06
15/06
18/06
19/06
20/06
21/06
22/06
25/06
26/06
27/06
28/06
29/06
02/07
03/07
04/07
05/07
06/07
09/07
10/07
11/07
12/07
13/07
16/07
17/07
18/07
19/07
20/07
23/07
24/07
25/07
26/07
27/07
30/07
31/07
01/08
02/08
03/08
Recherche sur les decals
Lecture du code des decals
Découverte de la Nel et de l'organisation du scene graph
Mise en place d'une application de test pour utiliser les fonctions de la NeL
Développement du noeud Decal
Développement de la sélection de faces
Développement du manager
Développement du clipping
Développement du support des coordonnées UV.
Validation
Débogage et test

.2 Arborescence de la librairie NeL

```
code
  + ryzom
  |  + ...
  |  + ...
  |  .
  |  .
  + nel
  |  + include
  |  |  + nel
  |  |  |  + 3d      *
  |  |  |  + logic
  |  |  |  + memory
  |  |  |  + misc    *
  |  |  |  + net
  |  |  |  + pacs
  |  |  |  + sound
  |  + src
  |  |  + 3d         *
  |  |  + logic
  |  |  + memory
  |  |  + misc       *
  |  |  + net
  |  |  + pacs
  |  |  + sound
  |  + tools
  |  |  + 3d
  |  |  + logic
  |  |  + memory
  |  |  + misc
  |  |  + net
  |  |  + pacs
  |  |  + sound




  * : répertoire de travail
```

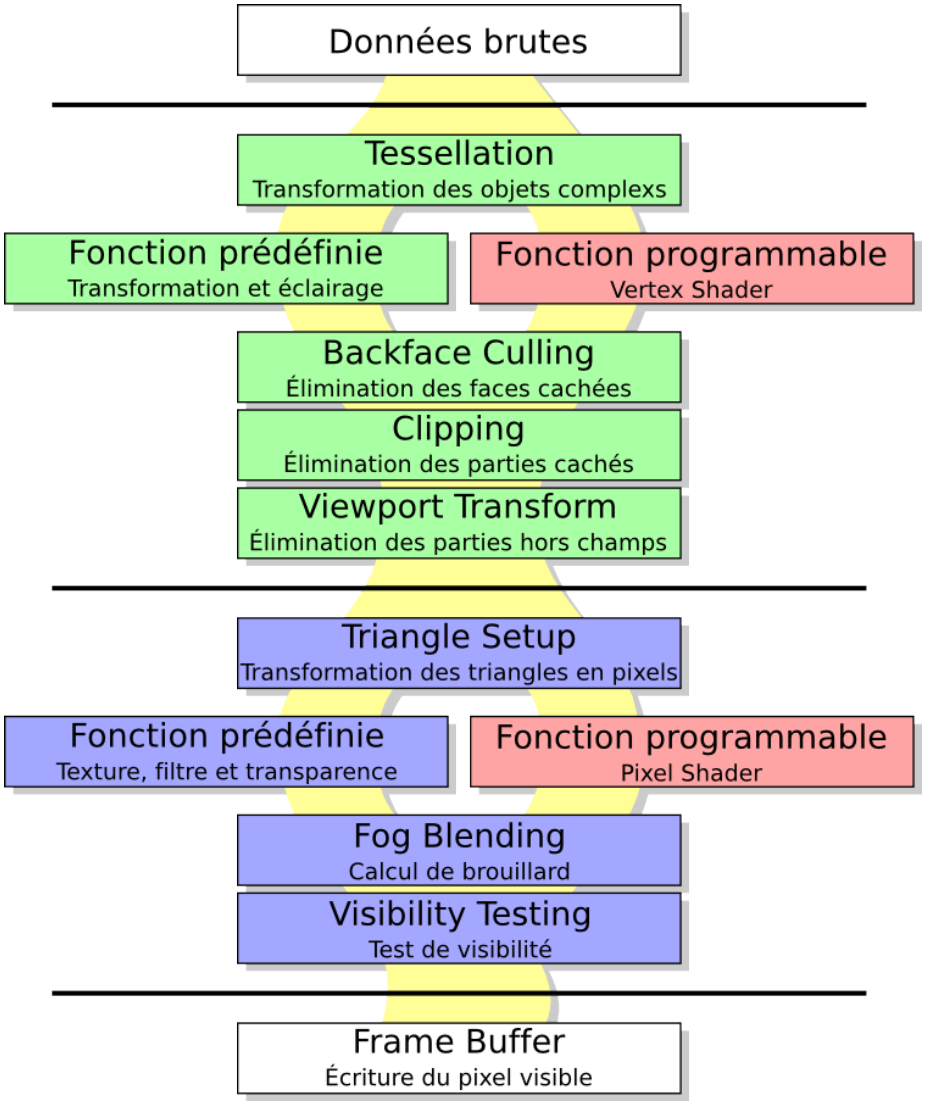
.3 Évènements CVS

date	project	content	link
17:09 on Aug 03	nel	Commit by **taranto** on WRK_IT2 :: **code**/nel/src/3d/ (4 files): Christopher - Decal: • Last commit, end of training period • known bugs : tile distance influence and first frame computing issue due to empty vertices tab	#
13:08 on Aug 02	nel	Commit by **taranto** on WRK_IT2 :: **code**/nel/src/3d/ (5 files): Decal - bug fix : • Clipping mode switch caused failure	#
16:06 on Jul 31	nel	Commit by **taranto** on WRK_IT2 :: **code**/nel/ (13 files in 3 dirs): Decal interface: • uv texture selection • allocate meme for material batching • set material id • set clipping mode	#
17:14 on Jul 30	nel	Commit by **taranto** on WRK_IT2 :: **code**/nel/src/3d/ (4 files): More efficient clipping with bottom and top planes, rotation ok	#
10:15 on Jul 30	nel	Commit by **taranto** on WRK_IT2 :: **code**/nel/src/3d/ (7 files): Decal offset and scale ok for object projection	#
16:58 on Jul 23	nel	Commit by **taranto** on WRK_IT2 :: **code**/nel/src/3d/ (7 files): Mask clipping ok, geometry clipping ok, traversal clipping ok	#
16:24 on Jul 18	nel	Commit by **taranto** on WRK_IT2 :: **code**/nel/ (10 files in 2 dirs): traverseClip, clipping geometrique traitant les UV, manager	#

17:09 on Jul 05	nel	Commit by **taranto** on WRK_IT2 :: **code**/nel/src/3d/ (7 files):
		`Decal :`
		`selection de face + clipping sur un objet -> ok`
			#

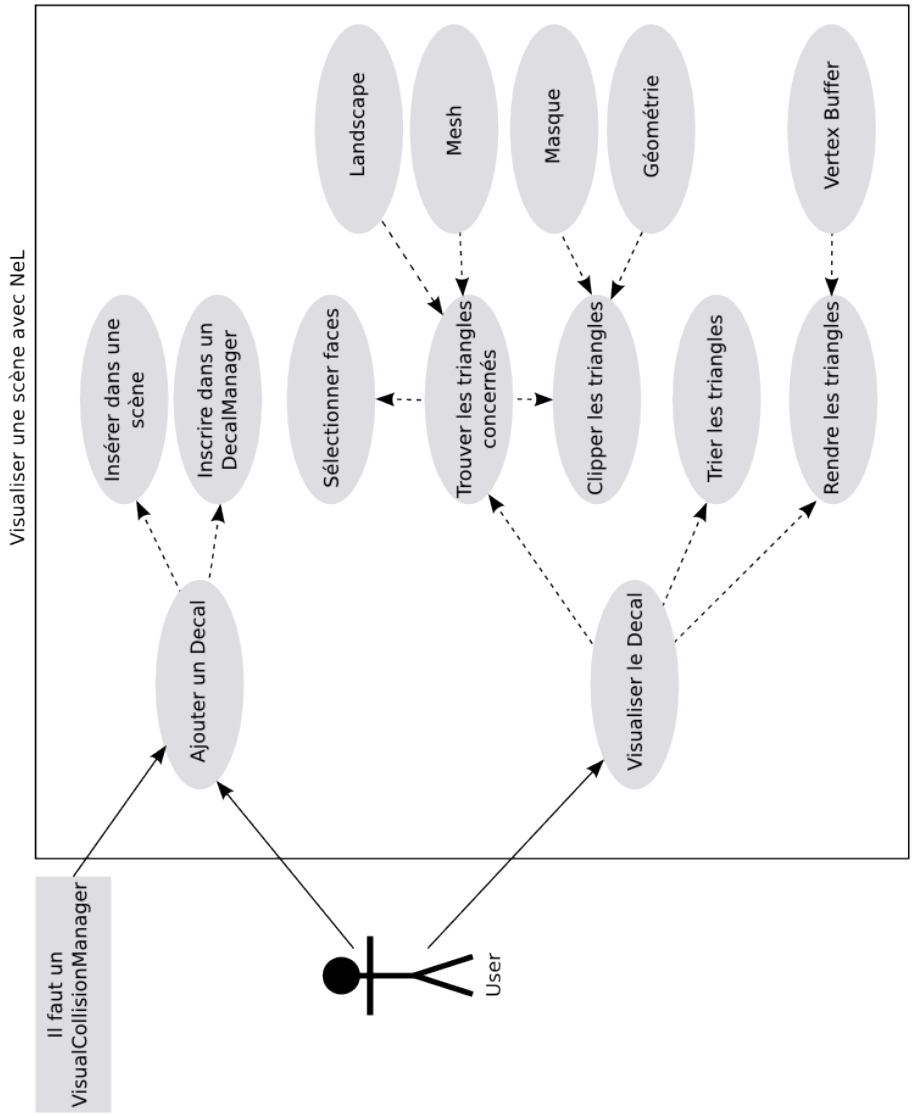
09:22 on Jul 02	nel	Commit by **taranto** on WRK_IT2 :: **code**/nel/ (21 files in 6 dirs): Decal by Christo	#

.4 Pipeline de rendu

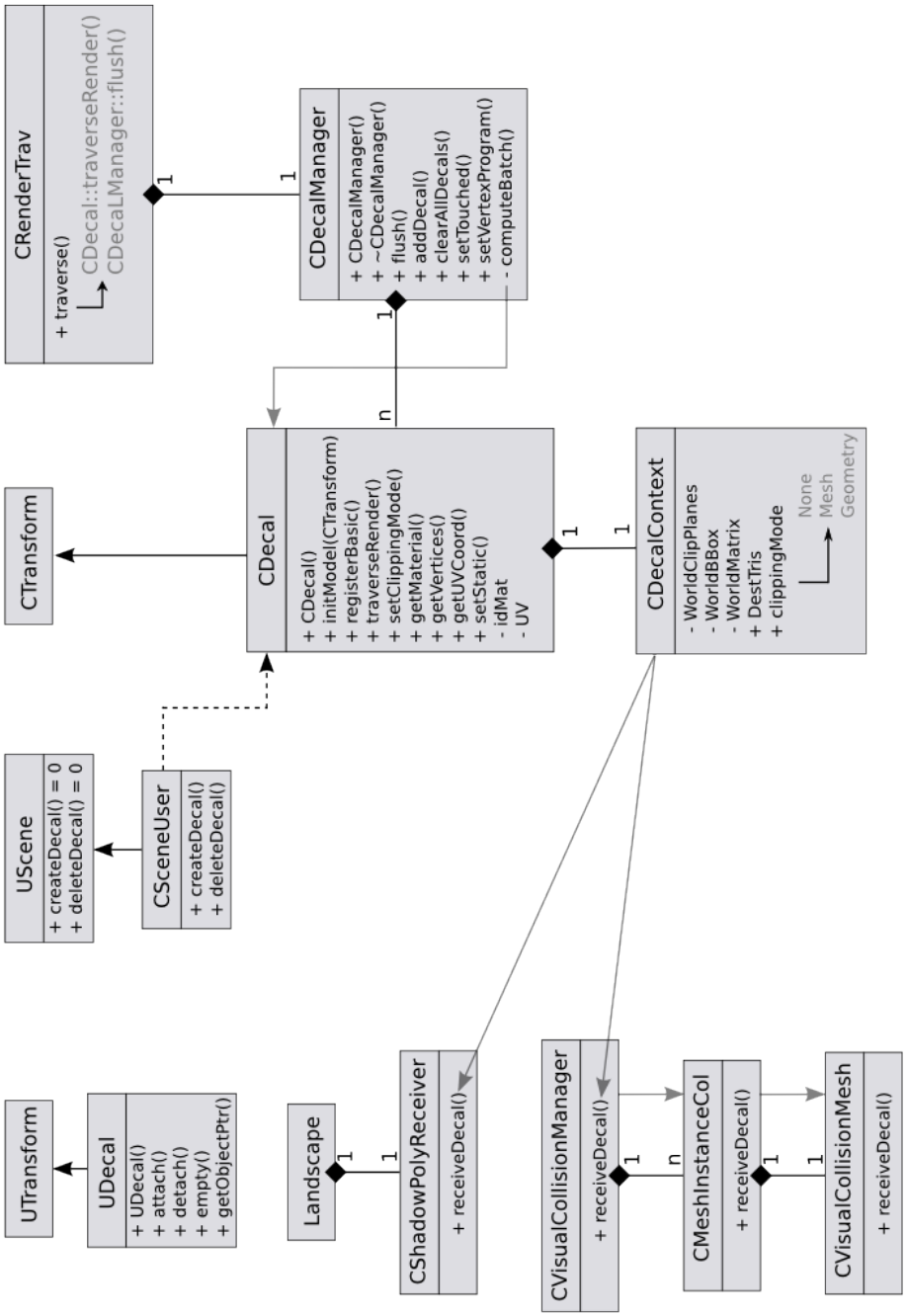


.5 Conception

.5.1 Use case



.5.2 Diagramme de classes



.6 Pseudo-code

.6.1 Clip traversal

```
fonction traverseClip :
{
  si dernière date de clip == date actuelle et que
     _Visible == vrai{
    return
  }
  _Visible = vrai
  dernière date de clip = date actuelle

  si la surface que représente le decal est inférieur à un pixel
    du point de vue de la caméra{
        _Visible = faux
  }
  sinon{
    float maxSide = taille du côté le plus grand de
      la zone de projection du decal

    BoundingSphere bs(centre du decal, rayon de longeur maxSide)

    vector<plan> worldPyramid
    scene->Camera->creerPyramideDeCamera(worldPyramid)

    pour chaque plan de worldPyramid{
      float d = plan courant * bs.Centre

      si d est supérieur à bs.Rayon
        _Visible = faux
        break
    }
  }
  si _Visible{
    ajouter ce decal à la liste des objets à faire passer dans le
      render traversal

    pour chacun des fils de ce noeud{
      appeler la fonction traverseClip
    }
  }
}
```

.6.2 Sélection de faces

```
fonction selection de face :
{
    sélectionner les triangles du quad grid selon
      la bounding box du decal

    pour chaque triangle sélectionné{
      pour chaque sommet{
        pour chaque plan de clipping de la bounding box{
          si plan courant * sommet courant > 0{
            /*propriété mathématique
              le signe de plan*sommet donne la position du point
              par rapport au plan*/
            sommet clippé
          }
        }
      }
      si au moins un point de non clippé{
        si pas de clipping{
          on ajoute les trois sommets au tableau de sortie
        }
        si vertex clipping{
          on ajoute les trois sommets dans un tableau provisoire
          on clippe les sommets du tableau provisoire
          on copie les sommets résultants dans le tableau
            de sortie
        }
        si mask clipping{
          on ajoute les trois sommets au tableau de sortie
        }
      }
    }
  }
}
```

.6.3 Manager, rendu

```
fonction rendu :
{
  pour chaque materiaux{
    count=0
    offset=0
    tant qu'il reste des decals{
      longueur=nombre de sommets pour le decal courant
      si longueur>0{
        incrémenter le decal courant
        offset=0
        continue
      }
      si count+longueur-offset > taille du array buffer{
        //pas assez de place
        copier{
          (taille du array buffer-count) sommets
          depuis la position (offset) du tableau de sommets
          vers la position (count) du array buffer
        }
        offset+=taille du array buffer-count
        rendre l'intégralité du array buffer
        count=0
      }
      sinon{
        //assez de place
        copier{
          (longueur-offset) sommets
          depuis la position (offset) du tableau de sommets
          vers la position (count) du array buffer
        }
        count+=longueur-offset
        incrémenter le decal courant
        offset=0
      }
    }
    si count>0{
      rendre les (count) sommets du array buffer
    }
  }
  vider le tableau de "decals à rendre"
}
```

Bibliographie

[Boy02] Régis Boyer. *L'Islande Médiévale*. Belles Lettres, 2002.

[DeL01] Mark DeLoura. *Game Programming Gems 2*. Charles River Media, 2001.

[McR] Tom McReynolds. Simple C code example of decaling, using stencil buffer.

[MO] Matt McClellan and Kipp Owens. *Alternatives to Using Z-Bias to Fix Z-Fighting issues*. Intel Software Network, Intel Software Network.

[SWND06] Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide*. Addison Wesley, 2006.

[Wik] Wikipédia. L'encyclopédie libre.