ROMAN BAYON
3A ENSPS : Acquisition et Traitement d'Images.

Mémoire de fin d'études.

Développement de techniques d'affichage non-photoréalistes efficaces pour un environnement de réalité virtuelle appliquée à l'archéologie.

Mots-clefs : rendu non-photoréaliste, environnement virtuel interactif, visualisation temps réel.

Encadrant INRIA : Dr G. Drettakis
Projet REVES.
INRIA Sophia-Antipolis

Tuteur ENSPS : Dr P. Tellier
LSIIT
ULP Strasbourg

1 septembre 2003

Table des matières

Table des figures

Liste des tableaux

Remerciements

Je tiens à remercier M. George Drettakis pour son encadrement et son soutien, ainsi que M. Pierre Tellier pour ses conseils.

Je remercie aussi l'ensemble des membres de l'équipe REVES, pour son aide, et surtout sa bonne humeur.

Résumé

La quête du réalisme a motivé l'essentiel des recherches depuis les débuts de l'informatique graphique. Toutefois une nouvelle aire de recherche se développe : le non-photoréalisme. Le but est d'obtenir des images artistiques, qui donnent au spectateur des informations que la simulation physique ne peut lui transmettre.

Lors de notre projet de fin d'étude au sein de l'équipe REVES, nous avons mis au point un outil logiciel complexe, ayant pour but d'accélerer les developpements informatiques. Nous avons ensuite utilisé cet outil avec des algorithmes de rendu non-photoréaliste afin de les porter sur un environnement de réalité virtuelle. L'outil s'avère fonctionnel, mais souffre de problèmes de performances pour nos applications, à savoir des scènes représentant des environnements de grande dimension. Nous avons donc implanté à nouveau ces algorithmes pour optimiser la chaîne de rendu. Enfin nous avons débuté une extension d'un de ces algorithmes pour améliorer l'effet esthétique sur les environnements virtuels.

Abstract

The quest for realism has motivated much of the history of rendering, the process of creating synthetic imagery with computers. However, a new trend is currently arising : non-photorealistic rendering, which consists in creating artistical images designed to bring the viewer more expressiveness rather than physical simulation.

During our internship among the REVES team, a complex software tool, aimed at accelerating software conception, has first been settled. Afterwards, this tool has been used to integrate non-photorealistic rendering algorithms into a virtual reality plateform. The functionnalities match our requirements but the software integration tool lacks efficiency when rendering large scenes. Hence, the algorithms have been reimplemented to optimize the rendering pipeline. Finally, one of these algorithms has been partially extended to improve the rendering quality for virtual reality applications.

Introduction

Les recherches en images de synthèse se sont beaucoup concentrées sur les images réalistes, ce qui est important pour un grand nombre d'applications, comme les simulations, ou les créations de maquettes numériques. Cependant grâce à des représentations plus artistiques ou "expressives", il est possible de focaliser l'attention du spectateur sur des objets d'intérêts, et donner une atmosphère particulière à la scène. Ainsi, les architectes et les archéologues sont intéressés par les techniques de rendu non-photoréaliste.

Notre projet de fin d'étude réalisé au sein de projet REVES de l'INRIA du 3 mars au 30 août 2003 s'inscrit dans cette thématique. Il s'agit d'apporter des outils de très haut niveau au sein de l'action ARCHEOS, consistant en une simulation virtuelle de la Tholos d'Argos.

L'objectif de notre projet de fin d'études est de mettre en place un outil permettant d'étendre le champ d'applications de RevesAPI (la plate-forme de développement de l'équipe) et d'implanter des algorithmes de rendu non-photoréaliste pour illustrer différentes hypothèses archéologiques (élaborées par Marcel Pierart et Patrick Marchetti) afin de les porter sur un environnement immersif.

Nous préciserons tout d'abord le contexte de ce projet et présenterons les environnements matériels et logiciels du projet REVES. Puis nous expliquerons notre réalisation de l'encapsulation de RevesAPI dans XP. Ensuite, nous détaillerons le fonctionnement des algorithmes de rendu non-photoréaliste que nous avons implantés.

1 Le contexte du projet de fin d'études

1.1 L'INRIA

1.1.1 Généralités

L'INRIA, Institut National de Recherche en Informatique et en Automatique placé sous la double tutelle des ministères de la recherche et de l'industrie, a pour vocation d'entreprendre des recherches fondamentales et appliquées dans les domaines des sciences et technologies de l'information et de la communication (STIC). L' institut assure également un fort transfert technologique en accordant une grande attention à la formation par la recherche, à la diffusion de l'information scientifique et technique, à la valorisation, à l'expertise et à la participation à des programmes internationaux. Jouant un rôle fédérateur au sein de la communauté scientifique de son domaine et au contact des acteurs industriels, l'INRIA est lui-même un acteur majeur dans le développement des STIC en France.

L'INRIA accueille dans ses 6 unités de recherche situées à Rocquencourt, Rennes, Sophia Antipolis, Grenoble, Nancy et Bordeaux, Lille, Saclay et sur d'autres sites à Paris, Marseille, Lyon et Metz, 3000 personnes dont 2500 scientifiques, issus d'organismes partenaires de l'INRIA (CNRS, universités, grandes écoles) qui travaillent dans près de 100 "projets" (ou équipes) de recherche communs.

L'INRIA développe de nombreux partenariats avec le monde industriel et favorise le transfert et la création d'entreprises (une soixantaine à ce jour) dans le domaine des STIC, notamment au travers de sa filiale INRIA-Transfert, promoteur de 4 fonds d'amorçage : I-Source 1 et 2 domaine des technologies de l'information et de la communication, C-Source (multimédia) et T-Source (télécommunications). L'INRIA est actif au sein d'instances de normalisation comme l'IETF, l'ISO ou le W3C dont il a été le pilote européen de 1995 à fin 2002. Enfin, l'institut entretient d'importantes relations internationales : en Europe, l'INRIA est fortement impliqué dans ERCIM, consortium qui regroupe 16 organismes de recherche et participe à 100 projets européens. Au niveau international, l'institut collabore avec de nombreuses institutions scientifiques (laboratoires de recherche conjoints à Moscou, Pékin, Mexico, etc, "équipes de recherche associées", programmes de formation et d'accueil)

Le budget est de 120 millions d'Euros HT, dont 1/4 provenant de contrats de recherche et de produits de valorisation. Au cours des années 2000 - 2003, dans le cadre de son contrat quadriennal, les moyens de l'institut ont été augmentés de près de 50%.

La stratégie de l'institut repose sur la combinaison étroite de l'excellence scientifique et du transfert technologique. Cinq défis scientifiques majeurs ont été identifiés :

- maîtriser l'infrastructure numérique
- concevoir les nouvelles applications exploitant le Web et les bases de données multimédias
- savoir produire des logiciels sûrs
- concevoir et maîtriser l'automatique des systèmes complexes
- combiner simulation et réalité virtuelle

Le projet REVES s'inscrit pleinement dans le thème de la réalité virtuelle.

1.1.2 Le projet REVES

L'objectif du projet REVES, créé en 2002, est de développer de nouveaux algorithmes afin d'améliorer et d'accélérer le processus de génération d'images de synthèse et de son spatialisé. Les algorithmes de rendu s'appliquent aux environnements virtuels ou augmentés (mélange virtuel et réel).

Les domaines d'applications visés sont, notamment, le patrimoine virtuel, le bâtiment et l'urbanisme, les jeux vidéo et l'audiovisuel. L'originalité du projet REVES réside dans l'étude et la modélisation cohérente du son et de l'image pour les environnements virtuels. Il est à présent largement reconnu que l'apport de son spatialisé en complément de graphiques 3D convaincants améliore sensiblement la sensation d'immersion et de présence dans les environnements virtuels.

REVES apporte des contributions au rendu d'environnements virtuels audio-visuels dans lesquels le son est une partie intégrante de l'expérience. A ce titre, REVES développe de nouveaux algorithmes temps réels pour la génération de son 3D, l'optimisation du rendu basée sur des critères perceptifs multi-modaux, la réverbération artificielle et la modélisation de l'acoustique de lieux virtuels, ainsi que la gestion combinée des ressources matérielles graphiques et audio.

Le projet REVES a développé une plate-forme commune de logiciels, REVESAPI, pour le rendu et les environnements virtuels sonorisés. Plusieurs modules sont en cours d'intégration dans cette plate-forme commune :

- ECLAIRES : un système de simulation d'éclairage complet pour les effets diffus et spéculaires (radiosité).
- SKEL : un système de calcul de visibilité exacte à base de calculs robustes
- PSM : des cartes d'ombres perspectives
- PBR : un système de rendu rapide par points
- AURELI : une librairie pour l'audio spatialisé

REVES est également en contact avec plusieurs sociétés qui s'intéressent à divers aspects de ces logiciels.

Dans le cadre des collaborations et des programmes financés, le projet REVES s'implique fortement dans les environnements virtuels et augmentés :

- Le projet européen IST "CREATE : Constructivist Mixed Reality for Design, Education, and Cultural Heritage ". Le but de ce projet est d'augmenter le réalisme des environnements virtuels et d'utiliser ces environnements pour un apprentissage"constructiviste"dans le domaine du patrimoine virtuel et pour l'urbanisme. Dans le cadre de ce projet, l'équipe REVES collabore avec le Centre Scientifique et Technique du Bâtiment (CSTB) pour les aspects d'urbanisme, la Fondation du Monde Hellénique (FHW) à Athènes pour le patrimoine virtuel et la société REALVIZ pour la capture des environnements réels. Le projet est coordonné par l'UCL à Londres. Dans le cadre de ce projet, REVES adapte les modules de la plate-forme REVESAPI pour une plate-forme basée sur les logiciels de réalité virtuelle CAVElib et Performer, intégrés dans le système d'interaction à base de scripts "XP"développés par la FHW.
- Le projet PREDIT "EVE-Visit"avec le CSTB. Le projet REVES a développé une simulation visuelle pour un système de " fenêtre virtuelle " qui sera utilisé par l'INRETS pour l'évaluation des nuisances sonores et visuelles.
- L'action de recherche coopérative (ARC) ARCHEOS est pilotée par REVES. Le but de cette action est de développer des outils pour le rendu non-photoréaliste (type "dessin" ou plus généralement de style artistique), dans des environnements immersifs. Les domaines d'application visés sont l'archéologie et l'architecture. L'action regroupe des historiens et archéologues, des spécialistes du rendu et de la réalité virtuelle.

Cette équipe est composée de deux chercheurs permanents, trois étudiants en thèse, et trois ingénieurs. De plus, l'INRIA Sophia Antipolis s'est dotée en Avril 2002 d'un plan de travail virtuel Barco Baron, qui est placé sous la responsabilité du projet REVES. La spécificité de ce dispositif est le fait d'être piloté par un PC avec une carte graphique standard, ce qui rend cette installation beaucoup plus accessible par rapport aux installations précédentes basées sur des calculateurs haut de gamme.

1.1.3 L'action de recherche : ARCHEOS

Malgré ses développements, le rendu non-photoréaliste a été très peu utilisé dans le contexte de systèmes d'environnements virtuels immersifs comme des CAVEs ou des RealityCenters. Deux problèmes majeurs expliquent ce manque.

- A cause de la nouveauté de ce domaine, les utilisateurs et les applications potentielles des environnements virtuels non-photoréalistes n'ont pas encore été identifiés. De plus , il n'y a pas eu beaucoup d'intérêt à procéder ainsi, car les aspects centraux de la recherche en réalité virtuelle, notamment la présence et l'immersion, ont été synonymes dans le passé à être crédible.

– Des problèmes technologiques subsistent pour le rendu non-photoréaliste dans un contexte d'environnement virtuel, notamment liés au rendu en stéréo pour certains de ces algorithmes, et à la rapidité de calcul des méthodes.

Cette action s'attaque à ces deux problèmes.

Un ensemble de participants ont été réunis autour de ce projet ; il compte :

– pour l'informatique graphique : les équipes REVES et iMAGIS (actuellement ARTIS).
– pour l'archéologie et l'histoire : l'ERGA (équipe de recherches sur la Grèce archaïque) et l'ENS.
– des architectes (ARIA)
– des spécialistes en utilisation des environnements virtuels pour la présentation du patrimoine virtuel, notamment dans un contexte éducatif (FHW).

Cette collaboration permettra d'abord de définir les scénarios d'application pour les visites virtuelles de sites archéologiques et, en particulier, elle permettra de déterminer à quel moment une représentation non-photoréaliste est souhaitable, dans un contexte de visite/balade virtuelle. L'interaction constante et rapprochée avec les archéologues, historiens, architectes et spécialistes des environnements virtuels pour le patrimoine est un élément clef pour la réussite de ce projet. L'application principale est la visite virtuelle d'Argos, qui est un projet déjà entamé par iMAGIS et ERGA.

En ce qui concerne les défis technologiques, plusieurs points sont abordés :

– développer des algorithmes pour l'affichage interactif de bâtiments antiques, dans un style à trait d'encre et d'aquarelle, ou dans un style qui sera jugé adapté pour le contenu, les sources disponibles et les utilisateurs visés (enfants, grand public ou experts).
– examiner les problèmes d'utilisation de niveaux de détails et de représentation appropriée pour l'illustration de différentes hypothèses sur une structure antique.

Dans un premier temps, la question de représentation de l'incertitude avec un niveau de détail approprié doit être étudiée. Puis, dans un deuxième temps, nous allons intégrer la visualisation des différentes périodes historiques d'un site en développant des algorithmes de visualisation permettant de transmettre à l'utilisateur l'information de variation dans le temps.

1.2 Problématique.

Le but du projet de fin d'étude est de mettre en place une plate-forme de réalité virtuelle qui permet l'affichage non photoréaliste en temps réel. Nous utilisons des composantes logicielles existantes (OpenGL, Performer, CAVElib) et nous implantons les algorithmes développés sur la plate-forme workbench de l'INRIA Sophia-Antipolis. Nous étudions plusieurs algorithmes de rendu non photoréaliste permettant différents types d'effets de rendu. Le stage se déroule dans le cadre de l'ARC (Action de Recherche Coopérative) ARCHEOS. Nous allons appliquer les algorithmes développés sur la mise en place d'un démonstrateur pour illustrer des hypothèses archéologiques pour la Tholos d'Argos (voir figure 1).

FIG. 1 – Rendu de Mental Ray pour Maya d'une reconstruction de la Tholos Argos.

Les tâches Notre contribution nous a été précisée à notre arrivée dans l'équipe. Il s'agit :

- d'étudier la faisabilité d'une encapsulation RevesAPI dans XP.
- de porter l'algorithme du papier dynamique sous RevesAPI et XP.
- d'implanter l'algorithme de rendu Raskar sous RevesAPI et XP.
- d'ajouter de nouvelles fonctionnalités aux algorithmes, si nécessaire.

Ces travaux ont une forte composante de "génie logiciel". De plus, les programmes à mettre en place doivent respecter les contraintes du temps réel. Ils doivent donc être suffisamment rapides pour interagir avec l'utilisateur.

La plannification Les principales étapes du projet sont présentées en figure 2.

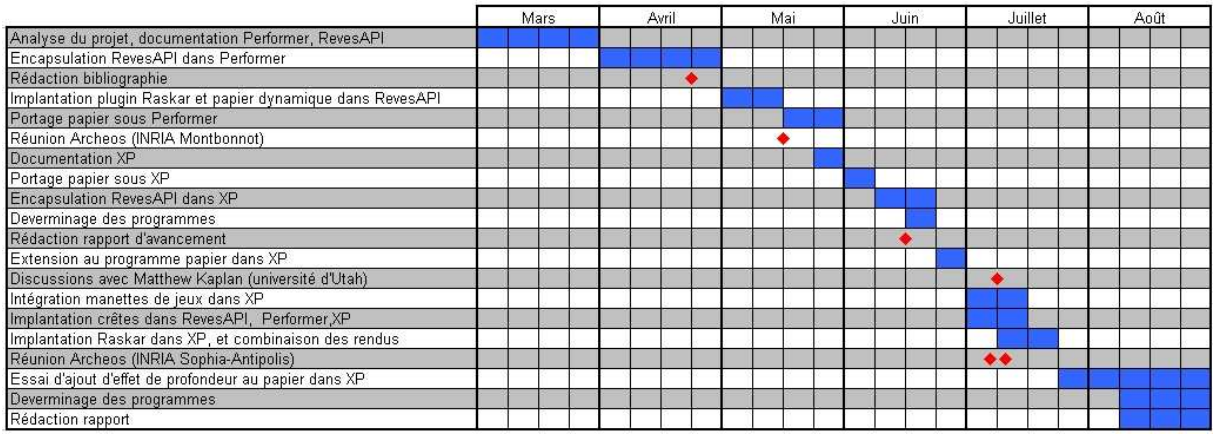

FIG. 2 – La plannification.

Ce planning effectif s'écarte assez peu du planning prévisionnel. Il a été possible de gagner un peu de temps sur la première phase du projet (wrapper, implantations dans RevesAPI), il en a été perdu un peu lors de l'implantation du papier dynamique.

2 La réalité virtuelle

2.1 Les environnement immersifs

Les environnements immersifs sont des ensembles complexes, composés d'une multitude de dispositifs. Ils comportent souvent plusieurs écrans, des lunettes stéréos, un appareil de suivi d'utilisateur (tracking), des outils de pointage d'objet etc. Le but de tels environnements est de "plonger" l'utilisateur dans un monde virtuel, entièrement géré par ordinateur.

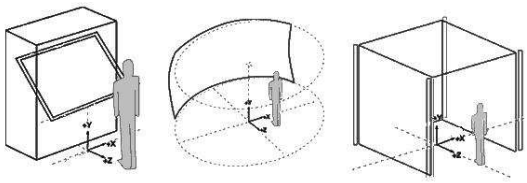

FIG. 3 – Sch´ema repr´esentant diff´erents environnements immersifs (table, mur immersif, cave).

Ces environnements se séparent en trois grandes catégories (voir figure 3) :
- les tables (un seul écran), c' est ce type de plate-forme que possède l'équipe REVES.
- les murs qui ne disposent pas forcément du suivi de l'utilisateur mais accueillent facilement plusieurs personnes.
- les caves qui entourent complétement l'utilisateur.

La figure 4 présente la plate-forme immersive de l'INRIA Sophia-Antipolis. L'utilisateur porte des lunettes stéréos auxquelles est rattaché un "mouchard" de suivi. Il porte en main un stylet de pointage qui lui permet d'agir sur la scène virtuelle. Fixé sur le workbench, nous trouvons l'émetteur stéréo qui agit sur les lunettes par transmission infrarouge, ainsi que le dispositif de mesure des positions (de l'utilisateur, de position et d'orientation du stylet). L'ensemble est piloté par un ordinateur standard sous Linux. Ces appareils permettent donc d'interagir dans le monde virtuel.

FIG. 4 – Plate-forme immersive (workbench) BARCO Baron.

Mais il ne faut pas oublier le point fort de cette machine, qui est la vision stéréoscopique. Grâce aux lunettes à cristaux liquides et à l'émetteur associé, l'utilisateur "voit en 3D" (i.e. il perçoit les objets en dehors du plan de l'écran.) En fait l'ordinateur de contrôle génère toujours deux images couplées, une pour

l'oeil droit et une pour l'oeil gauche. La machine workbench affiche alternativement chacune de ces images à la fréquence de 150 Hz. Les lunettes stéréos ne laisseront percevoir à chaque oeil que l'image qui lui est associée.

La figure 5 est un exemple de "stéréoimage". En observant attentivement nous remarquons un léger décalage entre les deux images (ex. muret entourant la tholos en bordure d'image). Ce décalage provient de l'écart de perception entre les deux yeux et permet au cerveau de reconstituer l'information de profondeur.

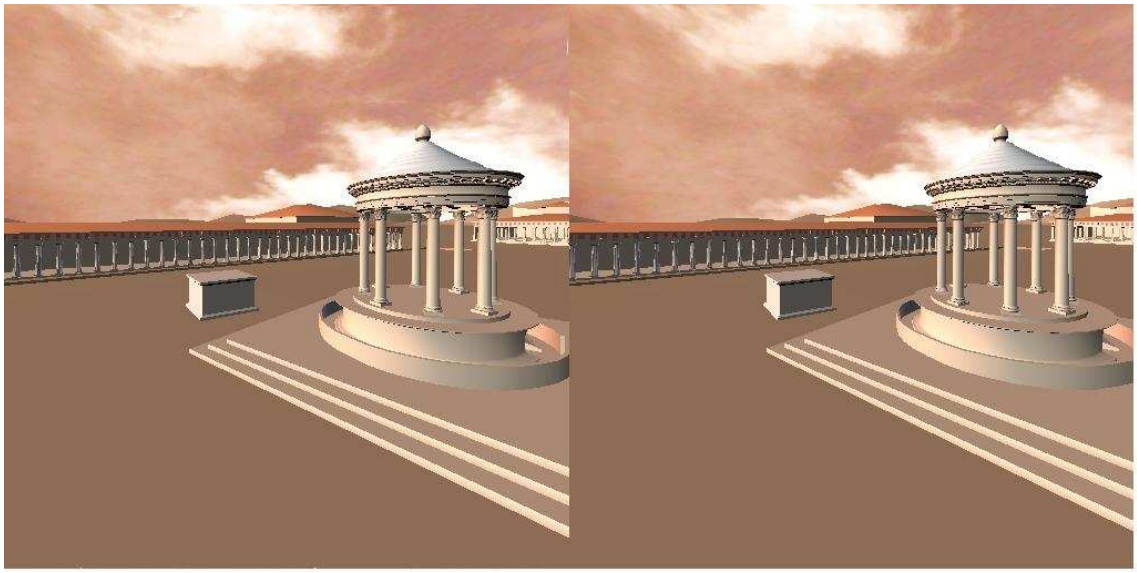

FIG. 5 – Exemple de "st´er´eoimage".

2.2 Les bibliothèques logicielles.

Les développements informatiques au sein du projet Reves se font toujours en C++, mais avec différentes librairies graphiques suivant la tâche d'application. Les travaux de recherche se font avec RevesAPI, tandis que les applications sur le workbench se font avec l'ensemble Performer/XP. Comme nous le montre la figure 6, ces librairies se basent toutes sur OpenGL. Elles sont, de plus, multi plates-formes (généralement Windows, Linux, Sgi).

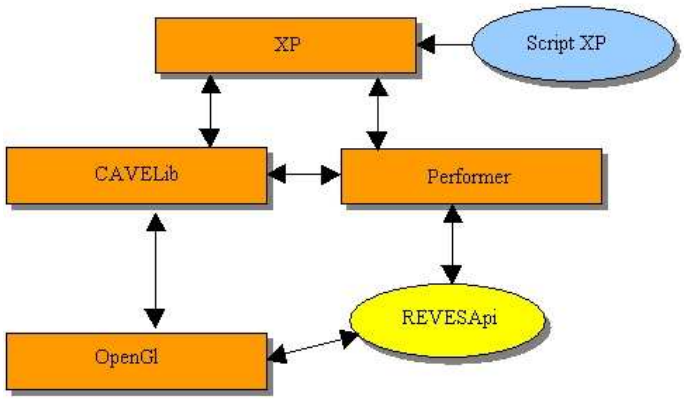

FIG. 6 – Les diff´erentes couches logicielles.

RevesAPI RevesAPI est la bibliothèque "maison" écrite par les membres de l'équipe au cours de leurs besoins de développement. La programmation se fait donc en parallèle, ce qui est possible grâce à un gestionnaire de version : CVS(Concurrent Versions System). Dans son utilisation "standard", RevesAPI est indépendante, et complétement "personnalisable".

OpenGL Performer OpenGL Performer fait partie d'une plate-forme globale conçue par Sgi (Silicon graphics incorporated) qui est depuis de nombreuses années une entreprise phare dans le domaine de l'informatique graphique. Performer ainsi que ses outils "satellites" est très répandu dans le monde de l'entreprise, essentiellement dans le domaine de la simulation. Les applications de Performer vont des simulations automobiles aux simulations médicales en passant par celles de défenses.

CAVELib CAVELib est développé par VRCO (Visualization : Interaction :Collaboration : Immersion). Cette société issue du domaine universitaire, à pour but de produire des applications permettant des interactions hommes-machines complexes. CAVELib est une interface mise à la disposition du programmeur pour ses applications immersives. Cette interface va s'occuper, à proprement parler, de l'aspect réalité virtuelle d'une application. Une même application peut donc fonctionner sur des environnnements immersifs différents. Il suffit juste que les configurations matérielles nécessaires à CAVELib aient été effectuées une fois pour toutes.

trackd trackd toujours développé par VRCO interagit de manière profonde avec CAVELib. C'est un petit démon qui va gérer les informations provenant des périphériques de réalité virtuelle. Il s'occupe de toutes sortes de périphériques, de la capture de mouvement au stylet etc.

XP XP est un environnement souple de description de scène. L'intérêt de XP est qu'il fournit une interface de haut niveau, pour la conception de scènes complexes. Un modeleur n'ayant quasiment aucune connaissance de programmation peut donc créer une scène interactive mettant en jeu des algorithmes complexes avec une grande simplicité.

3 Les outils logiciels

En début de projet, nous nous sommes familiarisés avec les différents outils de programmation utilisés au sein de l'équipe REVES. Notre travail nous a effectivement conduit à en utiliser plusieurs, et à étudier "en profondeur" leurs interactions. Après cette phase d'étude et d'apprentissage, nous avons réalisé une encapsulation de RevesAPI dans Performer.

3.1 Présentation

3.1.1 OpenGL Performer

Présentation OpenGL Performer est une bibliothèque de développement pour des applications 3D temps-réel. Les principaux champs d'application sont la simulation et la réalité virtuelle. OpenGL Performer est une surcouche d'OpenGL disposant de fonctionnalités et d'optimisations avancées. OpenGL Performer exploite donc au mieux les machines multi-processeurs, gère plusieurs caméras pour les systèmes de réalité virtuelle etc. Dans OpenGL Performer les données (la scène) sont codées sous forme de graphe : le scene-graphe (classe pfScene). Cette structure hiérarchique permet diverses optimisations.

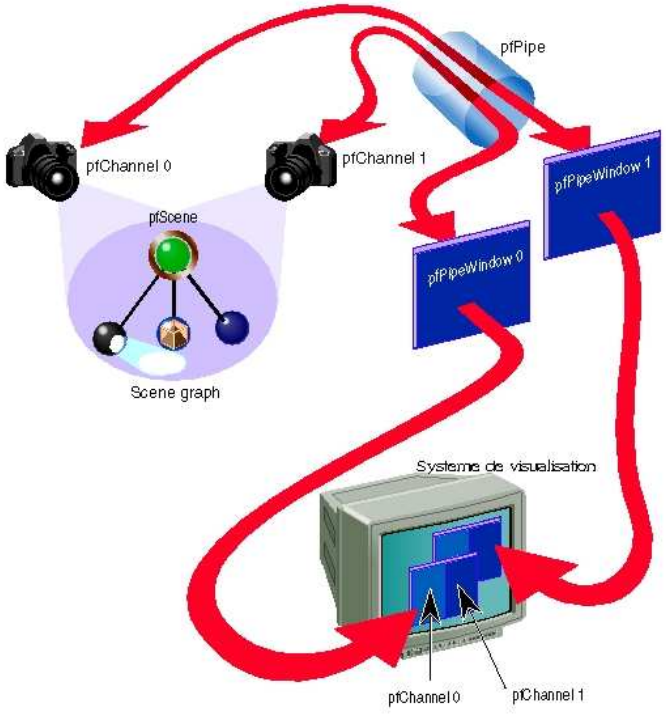

FIG. 7 – Performer : du scenegraphe à la visualisation.

Le scenegraphe est visualisé via une caméra (classe pfChannel), qui les retransmet pour le culling et l'affichage au pipeline de rendu (classe pfPipe). Ce fonctionnement est décrit en figure 7.

Le rendu de la scène se fait en trois étapes :

– APP - Actualise la position et le style de la géométrie, le point de vue et son orientation.
– CULL - Détermine la géométrie visible dans la scène, en tenant compte des ocultations.
– DRAW - Rend les objets visibles de la scène.

En utilisant Performer, son aspect ”surcouche OpenGL”, se fait fortement ressentir. Il est souvent assez simple de trouver la fonction OpenGL correspondant à un appel Performer. Il est donc possible de ”court-circuiter” des fonctions Performer, par des appels directs à OpenGL. Cette démarche demeure assez complexe, car il est alors difficile d'appréhender le déroulement précis du programme.

Performer autorise aussi la surcharge de certaines fonctions de bases (y compris au niveau du rendu : fonction *draw*). Il faut, pour cela, définir le *callback* adapté. Nous pouvons aussi définir des fonctions agissant avant et après les fonctions de base (exemple : ajout possible de fonctions *predraw* et *postdraw*, avant et après la fonction standard de rendu).

Quelques noeuds du scenegraphe Le tableau 1 présente les principaux noeuds du scenegraphe de Performer (et leurs classes associées).

pfScene	Racine du scenegraphe
pfNode	Noeud abstrait (père des autres noeuds)
pfGroup	Noeud de groupe, possède des enfants
pfSCS	Système de coordonnées statique
pfDCS	Système de coordonnées dynamique
pfGeode	Noeud de géométrie (abstrait)

TAB. 1 – Les principaux noeuds Performer.

Tous les noeuds héritent de **pfNode**. Les noeuds **pSCS** et**pfDCS** correspondent à des transformations de la scène. Les noeuds de type **pfGeode** contiennent, eux ,les informations de géométrie.

pfCAVELib CAVELib est un ensemble de librairies destiné au développement d'applications de réalités virtuelles. Il permet de gérer des environnements immersifs complexes, du suivi (des utilisateurs, des dispositifs de pointage) jusqu'à la visualisation.

pfCAVELib permet ainsi d'utiliser simplement Performer pour les applications immersives. Cette librairie crée les pfChannel et pfPipe adaptés à l' environnement de réalité virtuelle et utilise le noyau CAVELib ainsi que trackd pour gérer le suivi.

3.1.2 XP

Présentation A partir d'un fichier texte source (script), XP génère en paralèlle son propre scenegraphe, et un scenegraphe Performer. Ainsi, chaque pfNode aura un xpNode équivalent qui le contrôlera. Le rendu de la scène se fait alors grâce à pfCavelib. De plus, les xpNode, sont doté de mécanismes d'évènements et de communication de messages, qui permettent de gérer différents types d'interactions.

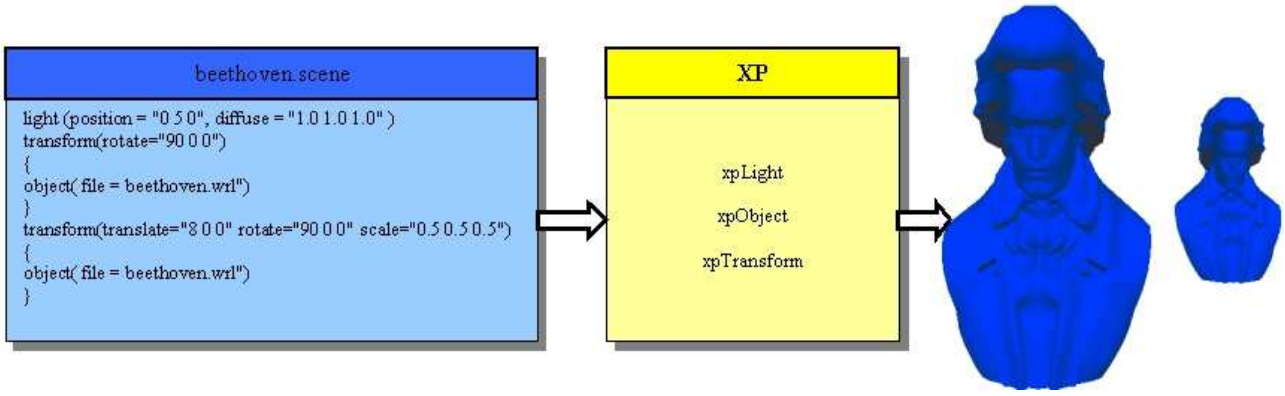

FIG. 8 – Du script XP à la scène.

La figure 8 montre un script XP. Le même objet est appelé plusieurs fois, avec des paramètres de transformations différents.

Intégration Performer dans XP XP est ”surchargeable”: il est en effet assez simple d'ajouter de nouveaux noeuds dans XP, à partir d'un noeud Performer.

La facon de procéder pour ajouter le noeud **monpfNoeud** dans XP est sommairement la suivante :

- Création de **monXPNoeud** héritant de **XPNode**.
- Surcharge des fonctions **XPNode : :parseOption** (pour récupérer les paramètres provenant du script) et **XPNode : :postInit** (pour agir sur les scenegraphes internes de XP).
- Déclaration **monXPNoeud** dans **xpWorld**

3.1.3 RevesAPI

RevesAPI est la plate-forme de développement commune du projet REVES. Elle met à disposition du programmeur un ensemble d'outils de base, afin d'accélérer le processus de développement, et lui permet aussi de partager les nouveaux modules avec l'ensemble de l'équipe.

RevesAPI distingue quatre grandes structures (voir figure 9) :

- **CrvData** : C'est la structure contenant les données, et les accesseurs aux données. Un scenegraphe **CrvScene** est un exemple de **CrvData** (i.e. **CrvScene** hérite de **CrvData**).
- **CrvModule** : Un module est constitué d'un ensembles d'algorithmes qui vont opérer sur les données.
- **CrvUI** : Une UI (User Interface) est l'interface utilisateur de l'application.
- **CrvController** : Un contrôleur est le lien entre une UI et un Module. Il sert à l'analyse des messages émis par l'UI, afin d'exécuter la fonction adéquate du Module.

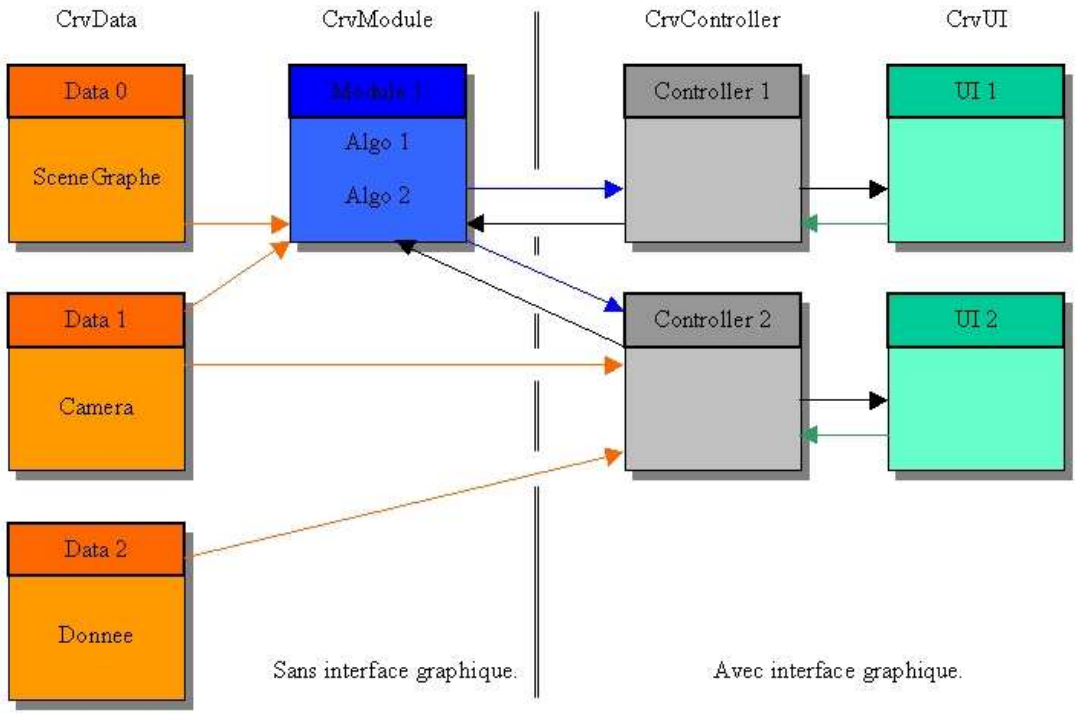

FIG. 9 – Structure de RevesAPI.

De plus, ces classes peuvent se transmettre des messages suivant un protocole précis. RevesAPI possède donc deux grands pôles : l'interface (UI, Controller) et l'algorithmique (Module, Data). Grâce à cette structure l'algorithmique produite dans les Modules est réutilisable.

Les patrons d'application RevesAPI met à disposition du programmeur des patrons (template) d'applications. Il existe, en effet, une classe **CtemplApp** qui crée une application complète capable de lire dans un fichier un modèle 3D, et de le visualiser. En héritant de cette classe, le programmeur peut, avec un minimum de codes (juste les surcharges adéquates), personnaliser et complexifier cette application de base. Ce patron d'application permet en outre l'utilisation de "plugin" dynamique de rendu construit sur la classe **CtemplRenderer**. Cette classe permet de créer n'importe quelle type de rendu à partir d'un scenegraphe RevesAPI. Un tel plugin peut être chargé dynamiquement au cours de l'exécution de l' *application patron*, via la classe **CdymodRenderSelectorModule**.

3.2 L'encapsulation de RevesAPI dans XP

Nous avons débuté notre travail par une étude pratique sur la possibilité de réalisation d'une encapsulation de RevesAPI dans XP (communément appellé wrapper). Nous voulons à terme que les travaux effectués avec RevesAPI, puissent être intégrés directement dans XP.

Grâce à cela, les techniques innovantes de l'équipe pourraient être utilisées dans XP sans aucune réécriture de code, d'où un gain de temps précieux. Ces techniques seraient donc directement placées dans un cadre applicatif (notammment le workbench) et utilisables aisément par le système de script.

L'intégration des classes Performer dans XP étant assez simple, nous nous concentrons tout d'abord sur le wrapping de RevesAPI dans Perfomer. Il faut donc créer une classe Performer permettant de :

– convertir un scenegraphe Performer en un scenegraphe RevesAPI,
– surcharger les fonctions *draw* de Performer pour exploiter le rendu RevesAPI.

Autrement dit, il faut remplacer une branche Performer par une branche RevesAPI équivalente, et ceci de manière transparente pour Performer (voir figure 10).

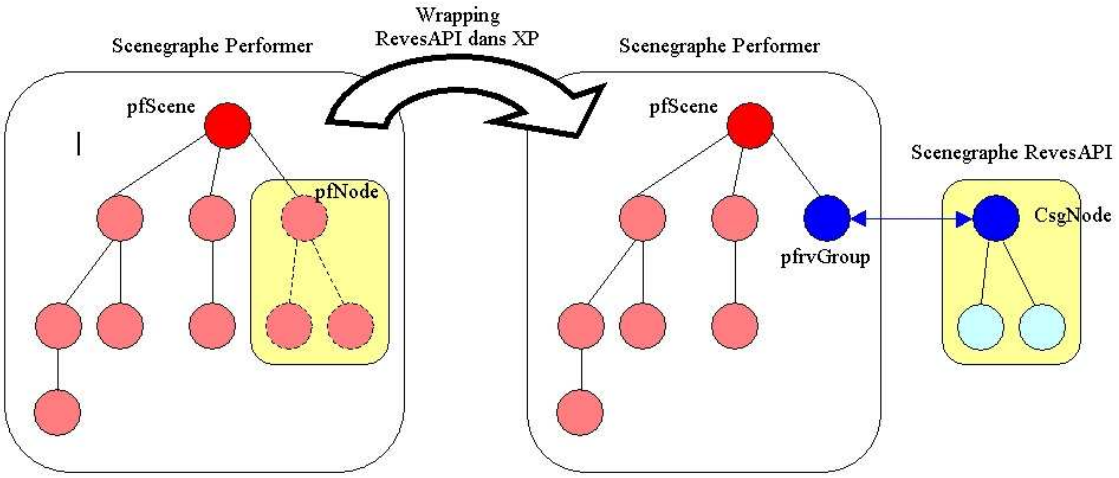

FIG. 10 – Wrapper : sch´ema de principe.

Ceci est théoriquement possible car RevesAPI et Performer sont tous les deux des surcouches OpenGL. Cependant les problèmes à résoudre sont nombreux :

– Compilation des programmes mêlant RevesAPI et Performer.
– Compatibilité de RevesAPI avec le multi-threading de Performer.
– Conversion de la géométrie et des matériaux Performer dans RevesAPI.
– Compatibilité des appels OpenGL dans Performer et RevesAPI lors du rendu.

3.2.1 La classe pfrvGroup

Nous avons alors developper une classe Performer pfrvGroup, héritant de pfGroup, et qui contient des fonctions membres adaptées (voir annexe C).

La construction Le constructeur de la classe accepte en entrée une chaîne de caractères contenant un chemin vers un plugin de rendu. Après avoir initialisé l'environnement RevesAPI, le plugin dynamique de rendu est chargé. Ainsi, notre classe contient un pointeur sur une classe de type **CtemplRenderer**. Enfin une fonction de prérendu (*preDraw*) est ajoutée à la classe (**static int pfrvGroup : :preDraw(pfTraverser *, void *)**). Nous redéfinissons la fonction de prérendu (*preDraw*) de la classe, car celle de rendu (*draw*) n'est pas accessible. La classe est alors encore extérieure au scenegraphe, et attend un appel à sa fonction membre **pfrvGroup : :rvWrap(pfGroup *pfgroupe)**. Cette fonction va effectuer l'encapsulation à proprement parler.

Les conversions Nous supposons ici avoir effectué l'appel suivant : *mongroupe* : :**rvWrap(***pfgroupe***)**

La première étape de la conversion est la recopie de la boîte englobante du noeud *pfgroupe* dans le noeud *mongroupe*. Ainsi le nouveau noeud sera géré convenablement par Performer notamment lors de la phase de culling. Ensuite vient la conversion à proprement parler.

Il y a trois grandes classes Performer à convertir : pfSCS, pfGeoSet et pfGeoState (enfant de pfGeode).

- Les noeuds de type pfSCS sont convertis en noeud de transformation de RevesAPI. Toutefois il est conseillé dans Performer d'effectuer un *"flattening"*. C'est un pré-processus qui consiste à appliquer les transformations des pfSCS directement à la géométrie des objets. Le parcours d'un *flatten*-scenegraphe est alors optimisé, car les matrices de transformation sont égales à l'identité, et peuvent donc êtres omises.
- Les noeuds pfGeoSet sont convertis en noeuds de géométrie RevesAPI. Les faces, sommets, normales sont transmis à RevesAPI mais pas les informations de texture, non pas par impossibilité technique mais par choix. Ce travail étant une étude de faisabilité, il était inutile de l'alourdir par des difficultés techniques supplémentaires. De plus le chargeur de fichier VRML utilisé par l'équipe ne génére que de la géométrie de type **PFGS_POLYS** (Perfomer polygone). Seul le traitement de cette géométrie est alors implanté.
- Les noeuds pfGeoState contenant les matériaux sont rattachés aux noeud pfGeoSet ; nous attachons donc dans RevesAPI le matériau adapté à chaque noeud de géométrie.

Finalement la branche Performer générée par *pfgroupe* est détruite, et les parents de *pfgroupe* se voient affectés *mongroupe*, comme nouveau fils.

Le rendu Lors du rendu du noeud **pfrvGroup**, Performer exécute tout d'abord la fonction de prérendu, puis celle de rendu (qui est imposée). Ayant détruit toute géométrie dans le sous-graphe Performer engendré par **pfrvGroup**, la fonction de rendu n'aura aucun effet. Nous exécutons donc le rendu du scenegraphe RevesAPI appartenant à la classe dans le preDraw en utilisant **CtemplRenderer : :Render(CsgNode *)** créé à la construction.

3.2.2 Paramétrage, compilation, exécution

La compilation de la classe **pfrvGroup** ne pose pas de réel problème dans Performer. Les librairies et header RevesAPI et Performer se revèlent parfaitement compatibles. Il suffit d'ajouter dans les Makefiles les liens entre librairies. Les appels OpenGL sont eux aussi compatibles entre eux, si nous prenons soin de sauvegarder les états GL avant les rendus.

Par contre RevesAPI ne supporte pas encore le multi-threading, alors que Performer l'utilise par défaut. Il faut donc forcer celui-ci à fonctionner en simple processus, ce qui se fait par l'appel à la fonction **pfMultiprocess(**PFMP_APPCULLDRAW **)**

Après avoir implanté des plugins dynamiques de rendu, cette encapsulation s'est montrée opérationnelle. Nous avons donc réussi à maîtriser la chaîne complexe de rendu graphique de Performer et le fonctionnement de RevesAPI, pour combiner les deux outils. La méthode d'utilisation de la classe **pfrvGroup** est donnée de façon plus complète en 5.2.

4 Le non-photoréalisme

Nous présentons ici quelques articles proposant diverses méthodes de rendu non-photoréaliste. Ils ne sont pas directement liés à notre application, et ne forment pas une étude exhaustive du non-photoréalisme. Toutefois ils permettent de se familiariser avec ce domaine, et ainsi d'appréhender certains problèmes qui peuvent se poser.

Il ne s'agit plus d'effectuer des modélisations précises de modèles physiques. Mais il faut comprendre les mécanismes de structuration et d'abstraction d'information induits par les représentations artistiques. Cela met en jeu des problèmes de cognition complexes.

4.1 Quelques éléments sur le rendu non réaliste...

Deussen et Strothotte [4] proposent une méthode de rendu d'arbre simulant une esquisse à la plume.

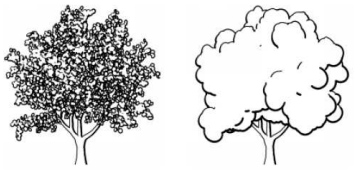

FIG. 11 – Rendu Deussen et Strothotte.

Les arbres sont modélisés de manière détaillée par leurs squelettes (troncs et branches) et leurs feuilles selon le système mis en place par Lintermann et Deussen [11]. Le but est alors de donner une représentation abstraite de l'arbre en seulement quelques contours. Le tronc est alors dessiné grâce à une variante de la méthode de Floyd Steinberg [6]. Pour les feuilles, les auteurs ont alors mis en place l'algorithme Depth Difference (différence de profondeur). Cet algorithme exploite le depth-buffer de l'image afin de repérer les primitives importantes pour la représentation du feuillage (voir figure 11).

Dans un autre article co-écrit par Durand et al., les auteurs ont développé une application originale [5], qui aide dynamiquement l'utilisateur à dessiner dans différents styles (voir figure 12). A partir d'une photo, l'utilisateur donne au système des consignes de bas niveau sur les grandes lignes du tracé, et des consignes de haut niveau sur le rendu voulu. L'application met alors en place ces consignes en temps réel. De plus, le

FIG. 12 – Illustration de la m´ethode Durand et al.

modèle de dessin utilisé est suffisamment souple pour simuler les effets de pinceau, gravage, fusain etc. Il est, en effet, capable de gérer différentes épaisseurs de traits et de pression de tracé.

Dans leur article [14], Mohr et Gleicher présentent une technique permettant de changer le style graphique d'applications 3D, qui met aussi en avant un nouveau problème. Ils interceptent les appels effectués par ces applications aux librairies de fonctions OpenGL, et remplacent les fonctions de dessins OpenGL par des fonctions qui leurs sont propres. Les auteurs utilisent tous d'abord un rendu en fil de fer, puis un rendu simulant basiquement le tracé au crayon noir, et enfin le tracé au crayon de couleur. L'inconvénient de cette approche est qu'elle met graphiquement en évidence la structure interne de la scène (tesselation des objets). Les primitives interceptées avant les appels OpenGL sont de trop bas niveau. La solution apportée, est de reconstituer la scène de façon plus globale en faisant du buffering. Alors, grâce à un raisonnement sur les normales des facettes, les discontinuités entre primitives sont réduites. Le rendu ainsi obtenu est de bonne qualité.

Enfin une équipe de Princeton [9] propose une technique de rendu en deux étapes. La première phase consiste à prendre une série de photographies d'un environnement réel ou virtuel, et de plaquer ces images sur un modèle 3D brut de l'environnement (voir figure 13). Alors des textures multi-résolutions sont

FIG. 13 – Rendu Klein et al.

générées à partir du modèle obtenu (appelé IBR pour Image Based Rendering) et de filtres de rendu non-réaliste. Le résultat de cette opération est un modèle NPIBR (Non Photorealistic Image Based Rendering). Cette phase de calcul importante est réalisée "hors-ligne". La deuxième phase est celle de rendu interactif. Il suffit alors d'exploiter le modèle NPIBR en reconstruisant une image correspondant au point de vue de l'utilisateur. L'utilisation des textures multi-résolutions (appelées *art-maps*) garantit la cohérence temporelle de l'effet "coup de crayon". De plus la qualité graphique est très bonne car l'essentiel du rendu est calculé hors-ligne.

B. Meier [13] présente une méthode pour rendre des animations avec un style peinture (voir figure 14).

FIG. 14 – Diff´erents types de rendus obtenus par B. Meier.

Dans un premier temps, B. Meier utilise la géométrie de la scène pour construire deux images de référence, et une image codant l'orientation des surfaces. Parallèlement, des particules sont placées sur les surfaces de la scène.

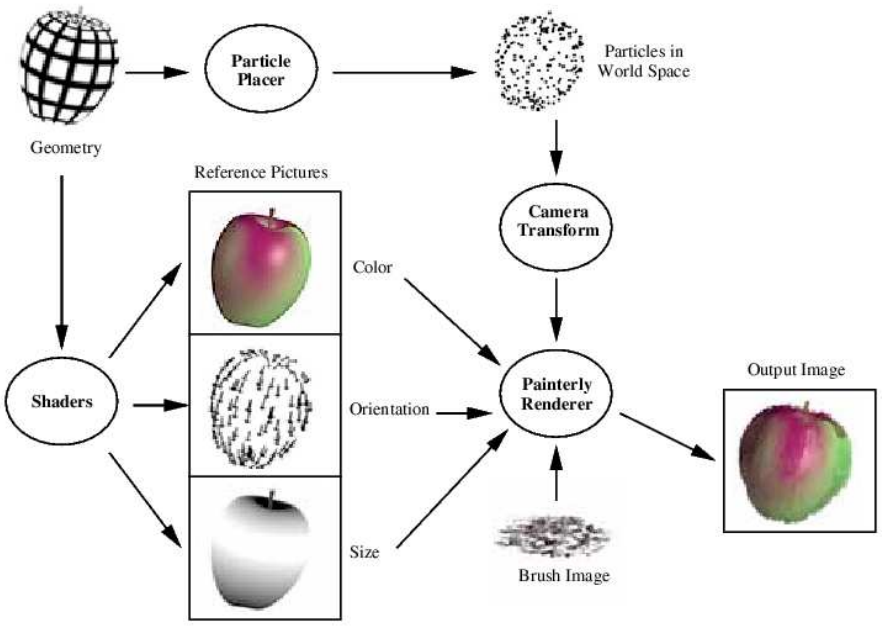

FIG. 15 – Pipeline de rendu B. Meier.

Alors, grâce aux particules et à l'orientation des surfaces, le moteur de rendu simule des coups de pinceaux adaptés aux images de référence (après une perturbation aléatoire des coups de pinceau, voir figure 15). De plus les particules ont des déplacements continus adaptés aux surfaces, ce qui garantit la cohérence temporelle des images.

Lake et al. se sont eux essayés avec succès au rendu style "cartoon" [10]. Ils présentent divers algorithme, de rendus et de détection de silhouette (voir figure 16).

FIG. 16 – Rendu cartoon et encre de Lake et al.

L'algorithme de détection de silhouettes est assez élémentaire, il consiste à générer une liste d'arêtes de silhouettes par l'étude exhaustive des *front* et *back-facing polygons*. Une fois ces arêtes connues, le dessin s'effectue soit avec un trait épais, soit avec un trait stylisé en fonction du voisinage de l'arête (plus précisément, de l'angle entre les segments voisins). Les silhouettes obtenues sont alors beaucoup plus plaisantes.

L'effet de rendu "cartoon" repose sur une quantification des couleurs possibles pour les matériaux, après un classique lissage de Gouraud. De cette manière, l'utilisateur retrouvera les couleurs "à-plat" des cartoons

traditionnels. Pour cela des textures unidimensionnelles sont précalculées ; elles contiennent les différentes couleurs autorisées pour les matériaux (typiquement matériel pur, matériel ombré). Ensuite les coordonnées de textures sont calculées à la volée, en fonction de la direction de la source lumineuse et de la normale au point. La méthode permet en jouant sur le nombre de niveau de quantification et sur les couleurs une grande variété de rendus. Cette méthode est par la suite étendue, pour simuler des dessins au crayon. La texture unidimensionnelle est alors utilisée pour indicer des textures (patchs) plus ou moins denses.

4.2 Papier dynamique, silhouettes et lignes de crêtes

Dans le cadre du projet ARCHEOS, nous avons été chargés d'étudier plus précisément l'algorithme de papier dynamique développé au sein d'ARTIS ; et des algorithmes de rendu par traits, à savoir le rendu de R. Raskar et le rendu des lignes de crêtes.

4.2.1 Le papier dynamique

Le but d'un canvas est de simuler une toile sur laquelle le peintre ferait son dessin. Ce problème est trivial dans des applications statiques (simples images), mais beaucoup moins en animation. Dans les animations non-photoréalistes sont souvent utilisés des fonds (canvas) statiques qui produisent un effet désagréable de *shower door*. Les objets de la scène glissent sur le fond, et le spectateur a l'impression de les observer à travers une sorte de milieu vitreux. Les studios Disney, par exemple, travaillent activement sur des techniques [3, 13] permettant d'éviter ce problème.

Le problème est effectivement complexe, car il consiste à simuler une toile (qui est en fait un plan de projection du monde) dans un environnement tri-dimensionnel dynamique. L'équipe ARTIS (INRIA Montbonnot) a proposé une méthode [2] permettant d'animer le canvas, pour donner une sensation de mouvement 3D et accroître ainsi la sensation d'immersion.

Présentation Cette méthode a été conçue pour effectuer des balades virtuelles. Elle est efficace pour des translations le long de l'axe optique, et pour des rotations autour des axes horizontaux et verticaux.

La méthode Le mouvement complexe 3D est approximé par des transformations purement 2D qui sont appliquées sur la texture de canvas. Le problème est décomposé en deux parties ; la première consiste à simuler les translations dans l'espace, la deuxième à simuler les mouvements de rotations de la caméra. Ainsi comme nous pouvons le remarquer sur la figure 17, les grains de la texture de fond (le papier dynamique) suivent le déplacement des objets. L'animation est cohérente temporellement.

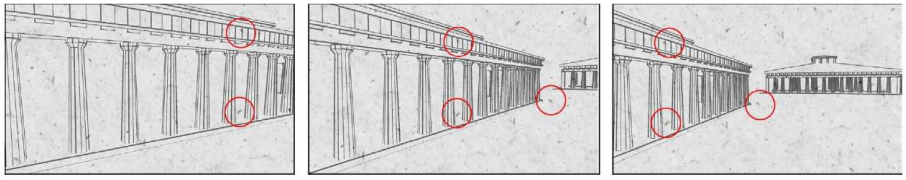

FIG. 17 – Balade virtuelle avec papier dynamique.

Translation Les translations le long de l'axe optique sont converties en un zoom 2D. La technique utilisée est astucieuse, elle consiste à combiner des représentations à différentes fréquences spatiales d'un même motif [1, 19] (principe de fonction auto-similaire). Plus précisément, nous définissons une octave de référence Ω_1 de fréquence F_1 et d'amplitude A_1. Les octaves Ω_i suivantes se déduisent par récurrence :

$F_{i+1} = 2 \times F_i$,$A_{i+1} = \frac{1}{2} \times A_i$ La composition (ou assemblage) des différentes textures se fait par une addition pondérée par le facteur de zoom $z \in [\frac{1}{2}, 1]$ (voir figure 18).

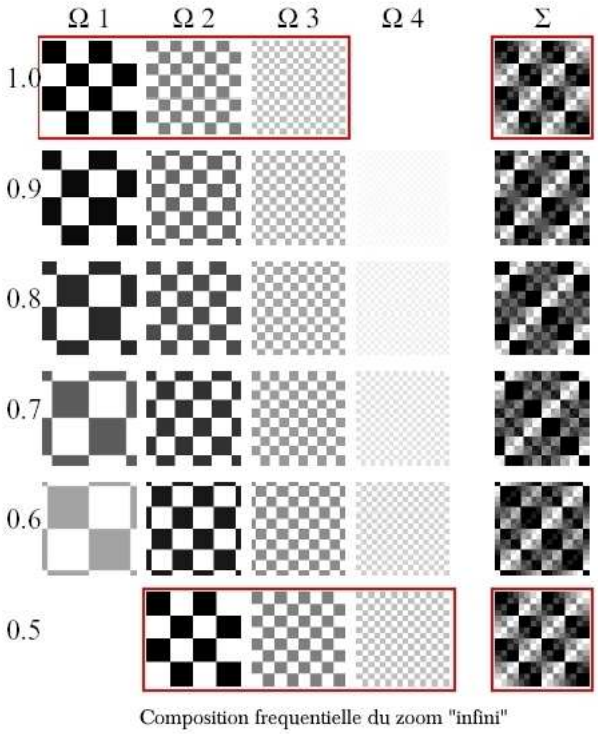

FIG. 18 – Illustration du zoom ”infini”.

Il est par nature difficile de représenter l’impression produite par l’animation. Mais en suivant les Σ_z consécutifs, nous avons l’impression que la texture zoome indéfiniment. Et en inversant le sens de parcours l’effet opposé est produit.

Toutefois cet effet de zoom infini ne résout pas les problèmes de rotation.

Rotation Pour rendre les effets de rotation de la caméra, le plan des textures est considéré comme tangent à une sphère de direction de vue (voir figure 19) avec, pour point de tangence, le centre de l’image. Localement, le plan sera considéré comme une approximation de la sphère. Ainsi pour le centre de l’image, le mouvement est rendu en faisant rouler la sphère de direction sur le plan sans glisser. Il est alors nécessaire d’ajouter une déformation afin que le flot optique défini par la texture soit plus réaliste. Ceci se fait grâce à une projection virtuelle de l’écran sur la sphère de direction.

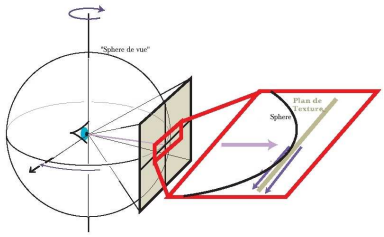

FIG. 19 – "La sphère de vue".

Cette méthode n'est donc pas parfaite car elle oblige à choisir une distance de placement du papier (distance de référence), et un glissement sera toujours présent aux bords de l'image.

Les pseudos-codes Nous trouverons en annexe A quelques détails sur l'implantation de la méthode. Mais nous pouvons la résumer de la facon suivante. Il s'agit tout d'abord de construire la texture (la composer) avec le facteur de zoom adapté, ce qui nécessite la connaissance des déplacements effectués entre l'image précédente et l'image actuelle. Ensuite la scène est rendue (avec un rendu non-photoréaliste à base de trait par exemple). Enfin le papier est affiché en semi-transparence, si nécessaire avec la déformation de rotation (le "*spherical warp*").

4.2.2 Les silhouettes

Les silhouettes jouent un rôle important dans le domaine du non-photoréalisme. Un rendu de silhouette simule bien les dessins par traits et permet de donner une information claire sur l'objet car épuré. Les silhouettes sont importantes dans la reconnaissance des objets, ce sont elles qui détachent les objets du fond de l'image.

Présentation La silhouette S d'un objet de forme quelconque est naturellement définie par l'ensemble des points de l'objet dont la normale est orthogonale au vecteur de vue. Ceci se traduit par :

$$S = \{P : 0 = \vec{n_i}.(p_i \vec{-} c)\}$$

Ou $\vec{n_i}$ et p_i sont respectivement la normale et la position du point P appartenant à l'objet et c la position du centre de projection.

Si cette définition est élémentaire, elle est néanmoins très difficilement exploitable dans la pratique.

En effet, pour les objets polygonaux rencontrés dans nos applications, les normales sont définies au mieux aux sommets des polygones. Ceci nous amène à redéfinir les silhouettes d'objets polygonaux de la façon suivante : Les *arêtes de silhouettes* d'un objet polygonal sont les arêtes de l'objet qui se partagent entre un *front-facing* et un *back-facing* polygone (voir figure 20). De plus la silhouette d'un objet est dépendante du point de vue d'observation, elle doit donc être recalculée pour chaque étape d'une animation.

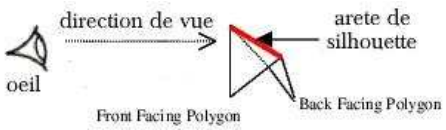

FIG. 20 – D´etection des silhouettes.

Les algorithmes de rendus de silhouettes peuvent alors se classer en trois grand types :

– algorithmes en espace image, qui opèrent sur les buffers images [4].
– algorithmes hybrides, qui opèrent partiellement dans l'espace objet, mais fournissent une silhouette 2D [15].
– algorithmes en espace-objet, qui opèrent entièrement dans l'espace 3D, et renvoient l'expression analytique des arêtes de silhouette [7, 12, 18].

Chaque type d'algorithme possède ses avantages et ses inconvénients. Les algorithmes de type "objet" sont plus précis, mais aussi plus lents, sans être forcément dénués de défauts [8]. Tandis que les algorithmes de type "ïmage", souvent plus simples à mettre en oeuvre, ne permettent pas d'obtenir une précision meilleure que le pixel.

Rendu Raskar Il a été décidé d'implanter l'algorithme de R. Raskar [17, 16]. Il appartient à la catégorie des algorithmes hybrides. Il est relativement rapide, car il nécessite seulement deux passes de rendu, et est donc idéal pour des applications temps réel.

La méthode repose sur l'utilisation du tampon de profondeur, qui permet de trouver les intersections des *front* et *back-facing* polygones adjacents.

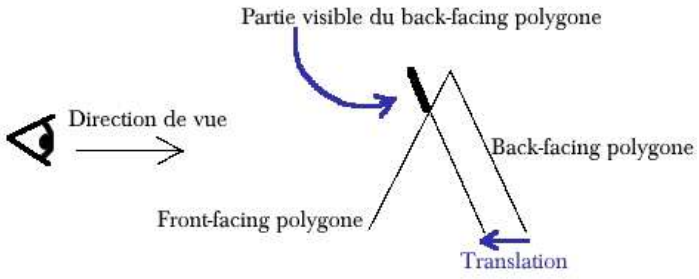

FIG. 21 – La m´ethode Raskar.

Lors de la première passe de calcul, seule la profondeur de la scène est rendue. Les objets sont alors déplacés vers le point de vue, et la seconde passe de rendu est appliquée seulement sur les *back-facing* polygones (voir figure 21). Cette méthode produit des épaisseurs de lignes différentes en fonction de l'angle entre les faces adjacentes. Ceci donne au rendu une agréable impression de dessin manuel.

Le pseudo-code est le suivant :

Algorithme `Rendu Raskar`
Entrée : Scenegraphe S .
Sortie : Silhouette en précision pixel.
Début

```
CullFace(BACK) ; // Masque les faces arrières.
PolygonMode(FRONT) ; // Active la face avant des polygones.
ColorMask(FALSE) ; // N'écrit aucune couleur( mais écriture dans le z-buffer).
    Render(S) ; // Traverse le scenegraphe(rendu de la scène avec le statut actuel)
ColorMask(TRUE) ; // Restaure l'affichage des couleurs.
PolygonMode(BACK) ; // Active les faces arrières des polygones.
PolygonOffset(α) ; //Ajoute un offset au z-buffer
```

CullFace(FRONT) ; // *Masque les faces avant.*
Color(BLACK) ; // *Change le matériau des objets(couleur noire)* .
Render(S) ; // *Traverse le scenegraphe(rendu de la scène avec le statut actuel)*
Fin.
Nous pouvons observer en figure 22 des rendus Raskar.

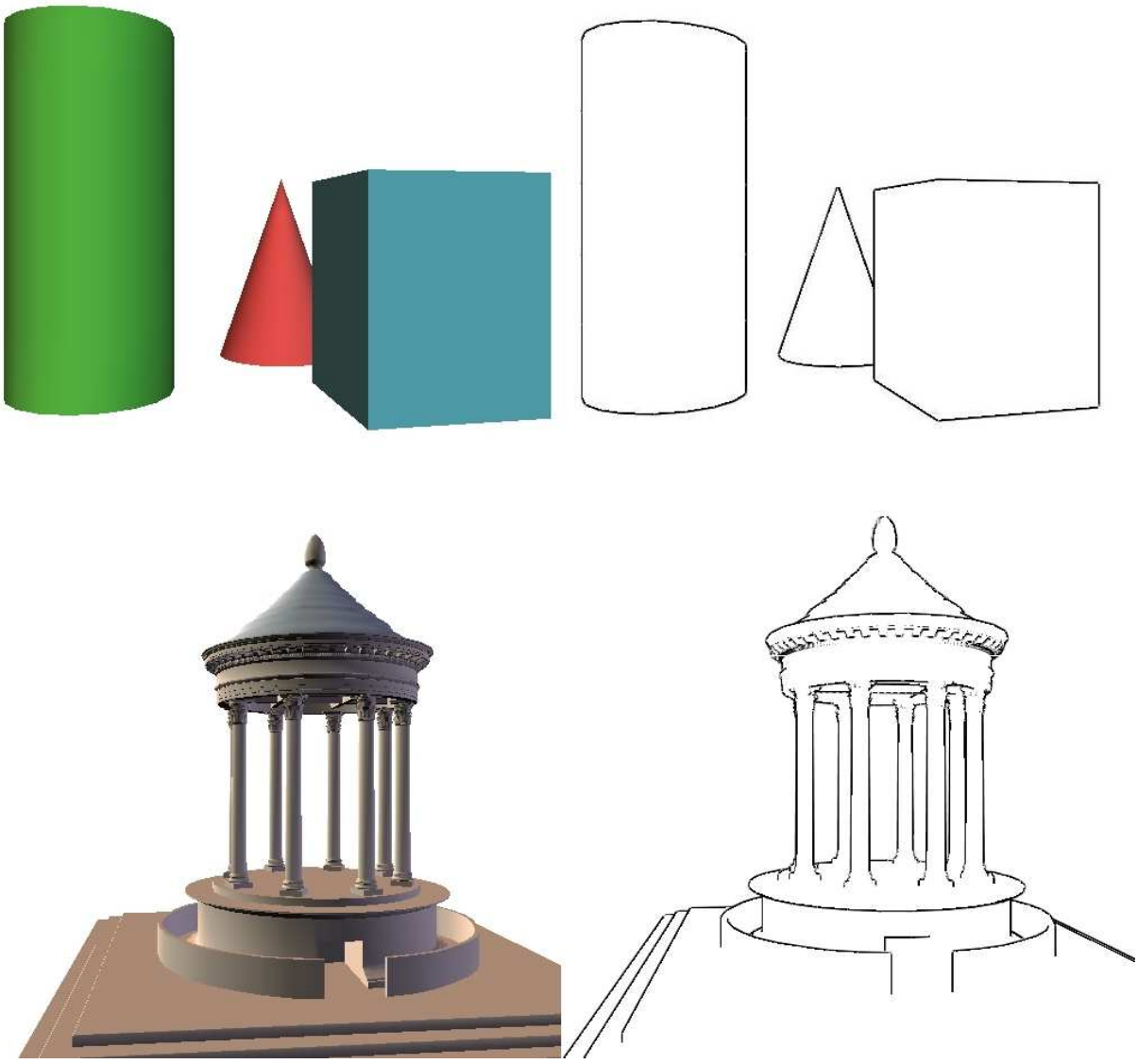

FIG. 22 – Quelques rendus Raskar.

Nous notons quelques imprécisions sur les silhouettes. Elles sont dues à la quantification du tampon de profondeur, et dépendent des paramètres de translations choisis.

4.2.3 Les lignes de crêtes.

Présentation Lorsque deux polygones adjacents forment un angle dépassant un certain seuil, ils définissent une ligne de crête, qui est l'arête commune aux deux polygone (arête rouge sur la figure 23).

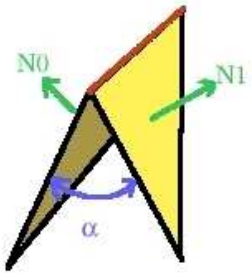

FIG. 23 – Lignes de cr^etes.

Pour des objets non déformables comme ceux que nous utilisons en pratique, les lignes de crêtes sont constantes au cours du temps. Nous pouvons donc les pré-calculer.

Nous exploitons les informations de normales pour les calculs des lignes de crêtes. Soient deux faces adjacentes F_0 et F_1 avec leurs normales respectives $\vec{N_0}$ et $\vec{N_1}$ et le seuil d'angle $\alpha \in [0; \frac{\pi}{2}]$; alors F_0 et F_1 produiront une ligne de crête si :

$$|\vec{N_0}.\vec{N_1}| \leq \cos(\alpha)$$

Nous devons aussi mettre en valeur les arêtes de bord (”border edges”), qui sont les arêtes reliées à un seul polygone.

Algorithme Nous avons élaboré un algorithme simple pour la recherche des lignes de crêtes (voir annexe B). Il consiste tout d'abord à effectuer un double parcours de l'ensemble des faces du modèle pour reconstruire la connectivité des faces (i.e. retrouver les faces adjacentes). Suivant la structure de données cette phase peut être facultative. Ensuite, pour deux faces adjacentes, nous testons si elles possèdent des normales à leurs sommets (objets lissés), ou si les normales sont définies pour les faces. Si les normales sont définies pour les faces, nous testons si l'angle entre les faces dépasse le seuil imposé.

Même si cet algorithme est relativement sommaire, il reste néanmoins efficace. Comme il s'agit d'un calcul effectué avant la phase de rendu et d'animation, la rapidité n'est pas un facteur critique.

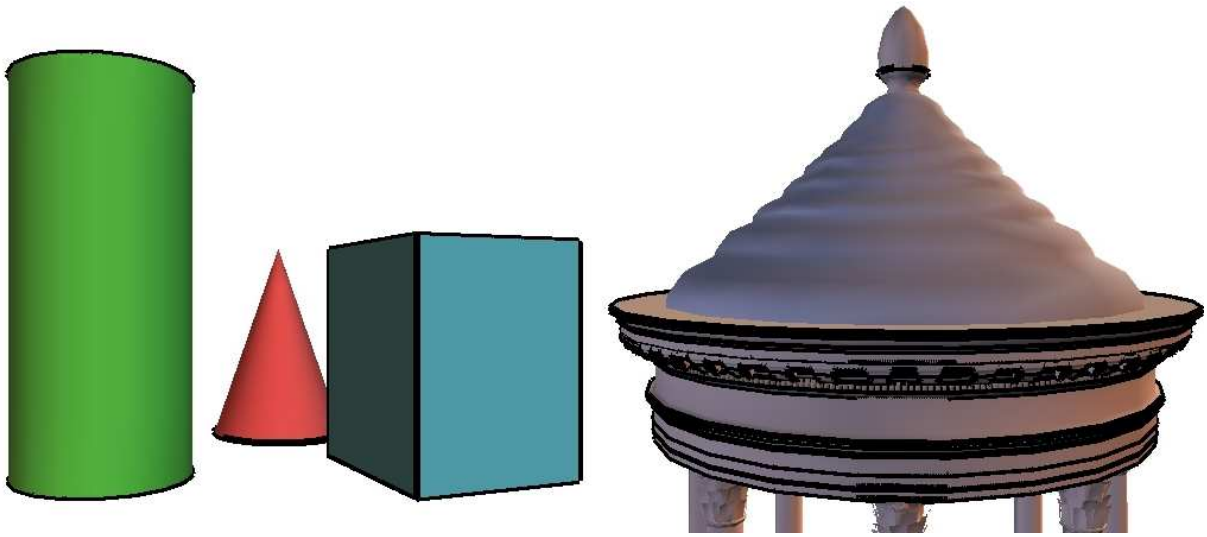

FIG. 24 – Quelques lignes de crˆetes (angle=15 degr´es).

Résultats Nous remarquons sur les images de la figure 24 deux inconvénients des lignes de crêtes. Les objets lissés (cône, cylindres...) ne comportent pas de lignes de crêtes, et leur silhouette ne sont pas mises en valeur, ce qui rompt la sensation de dessin. Et comme nous le voyons sur la partie supérieure de la Tholos, les lignes de crêtes provoquent rapidement des effets de saturation dès que les modèles sont détaillés. Il est donc (comme nous le verrons en 5.3.3) intéressant de combiner les lignes de crêtes avec le rendu Raskar, car ces deux modes sont complémentaires.

5 L'implantation des algorithmes

Après avoir assimilé les algorithmes du papier dynamique, du rendu Raskar et des lignes de crêtes, nous les avons implantés dans RevesAPI, afin de tester notre encapsulation RevesAPI/XP. Ensuite pour des raisons de rapidité d'exécution, nous les avons reprogrammés dans Performer/XP puis avons ajouté une extension à l'algorithme du papier dynamique.

5.1 L'implantation dans RevesAPI

5.1.1 Les lignes de crêtes

RevesAPI dispose d'une classe **CrdDisplayList**, qui à partir du scenegraphe, génére une liste d'affichages OpenGL (display list). Les cartes graphiques actuelles profitent de ces listes d'affichages pour optimiser et accélérer le rendu.

Les lignes de crêtes ont donc été implantées dans une classe **CrdDisplayListCrease** héritant de **CrdDisplayList**. Le mode d'utilisation de cet outil est donné en annexe D. Cette classe permet de modifier la construction de la liste d'affichage classique, pour y intégrer directement les lignes de crêtes (voir figure 25).

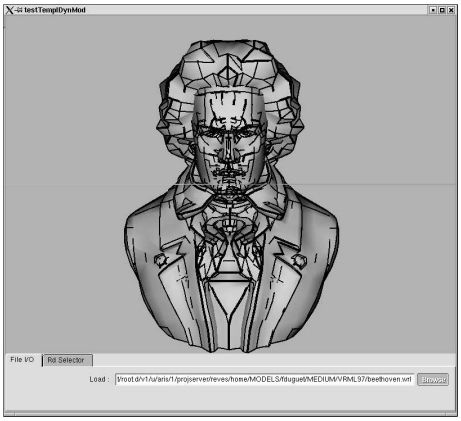

FIG. 25 – Le rendu de crˆetes sous RevesAPI (angle=15 degr´es).

5.1.2 Le rendu Raskar

Nous avons tout d'abord implanté le rendu de silhouette Raskar dans RevesAPI (voir figure 26). Pour ce faire, nous utilisons les plugins dynamiques de rendu. Il suffit donc de créer une nouvelle classe **CraskarRenderer** héritant de **CtemplRenderer**, et de surcharger les deux méthodes **Render** conformément à l'algorithme décrit en 4.2.2.

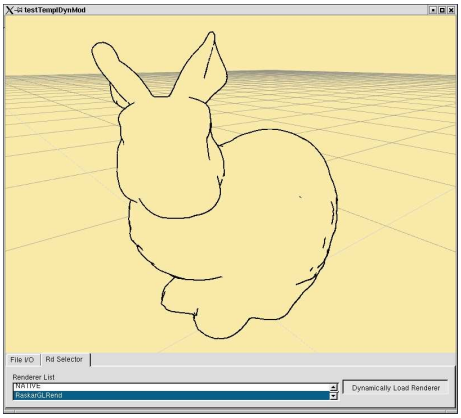

FIG. 26 – Le rendu Raskar sous RevesAPI.

5.1.3 Le papier dynamique

Malgré la complexité de l'algorithme du papier dynamique, l'intégration dans RevesAPI s'est faite relativement rapidement, car nous disposions d'éléments du code source de M. Cunzi.

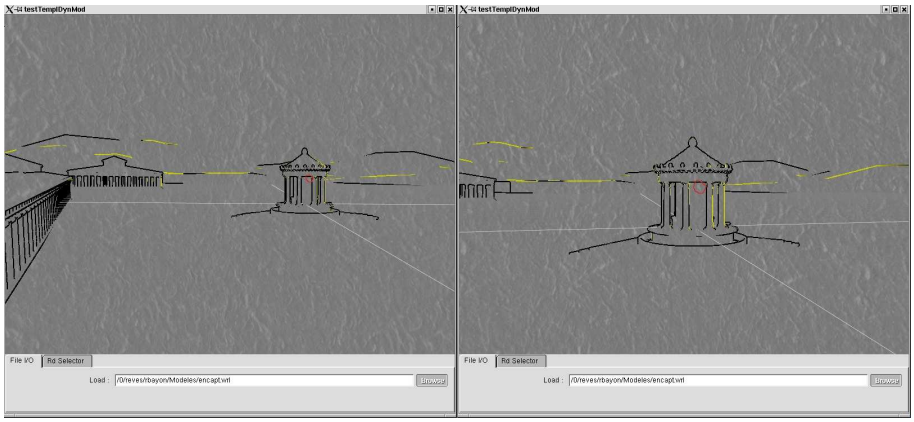

FIG. 27 – Le papier dynamique sous RevesAPI.

Avec la figure 27, nous pouvons valider l'algorithme. Cependant, pour pouvoir accomplir une balade virtuelle, nous avons dû ajouter une fonctionnalité à la caméra de RevesAPI. Cet ajout est la fonction membre **RotateAroundEye** dans le module **CmodCameraControl**. Cette fonction permet à la caméra de "tourner sur elle-même".

5.2 L'intégration RevesAPI dans XP

Une fois l'encapsulation en place dans Performer grâce à la classe **pfrvGroup**, nous l'intégrons dans XP en créant la classe **xpRvObject**. Nous obtenons alors l'objet XP *rvobject* (voir figure 28).

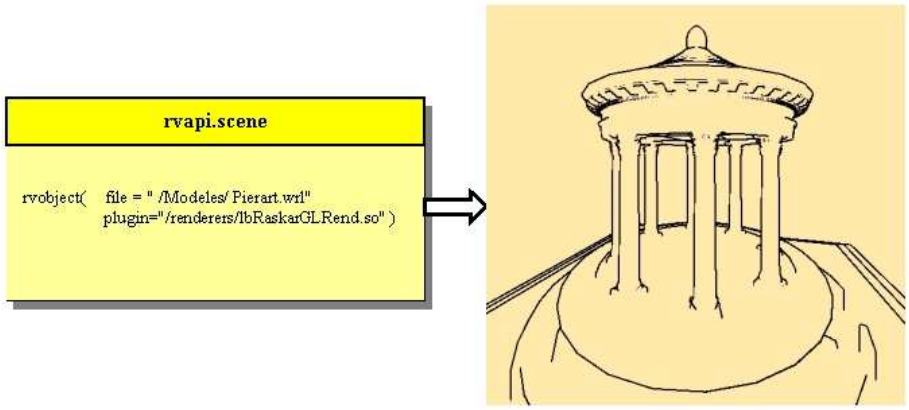

FIG. 28 – Script XP pour objet RevesAPI.

Cet objet accepte deux arguments.
– *file* : qui est le chemin sur un objet 3D (la branche du scenegraphe Performer que l'on va convertir).
– *plugin* : qui est le chemin sur un plugin de rendu RevesAPI.

5.3 L'implantation dans XP

5.3.1 Le papier dynamique

Nous avons tout de suite re-implanté l'algorithme du papier dynamique dans XP. En effet le système "composition de texture, rendu, affichage texture" ainsi que la transmission des informations de position rendent très compliquée l'utilisation de plugins RevesAPI. Cet algorithme est en outre très spécifique. Il n'y a pas grand intérêt ici à se servir de l'encapsulation RevesAPI-XP. Le portage de l'algorithme se montre déjà suffisamment délicat. Comme toujours nous portons le papier dynamique dans Performer d'abord, puis dans XP.

Le principal problème du papier est qu'il n'est pas un objet à part entière. Il ne peut appartenir au scenegraphe Perfomer. Comme le montre la figure 29, nous projetons le papier sur un plan occupant entièrement le champ de vue. Ceci se fait donc par des appels OpenGL. Nous sommes alors confrontés au problème de

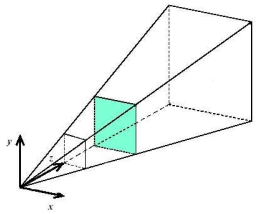

FIG. 29 – Le papier dynamique.

changement de repère imposé par Performer par rapport à OpenGL.

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}_{pf} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}_{GL}$$

De plus dans Performer la caméra reste fixe, alors qu'elle est mobile dans XP. Enfin il est nécessaire d'enregistrer les textures de papier au sein de XP. Ce programme est donc assez complexe à mettre en place et nécessite une re-écriture quasi complète du code d'origine.

Résultats sur l'environnement virtuel Les tests en monovision ont permis de valider cette implantation. Puis nous avons effectué des tests sur l'environnement immersif. Comme nous pouvions nous y attendre le résultat est assez étrange. Malgré plusieurs discussions à ce sujet, (avec Matthew Kaplan de l'université d'Utah, Pascal Barla, François-Xavier Sillon, Joëlle Thollot d'ARTIS ...), nous n'avons pas réussi à préciser exactement l'effet perceptif obtenu. En tous cas, le papier sur la plate-forme Barcon Baron, provoque un nouvel effet de "shower door". De plus au niveau de la profondeur il est perçu dans le plan de l'écran. Nous avons donc entrepris d'effectuer une extension à cet algorithme exploitant les spécificités de la vision stéréo.

5.3.2 Une extension de l'algorithme du papier dynamique

Lors de la réunion ARCHEOS de mi-juillet 2003, nous avons essayé de mettre au point une extension au papier dynamique. Nous avons ainsi profité de l'expérience de Joëlle Thollot, co-auteur de la méthode originelle. Notre but est d'exploiter la spécificité de la vision stéréo pour donner une notion de profondeur au papier. Nous voulons qu'il donne une sensation d'uniformité à la scène, à l'image d'une toile de papier. Il faut éviter l'effet "shower door" et l'effet "tapisserie" (à savoir une texture indépendante plaquée sur chaque objet). Pour cela nous créons un maillage que nous déformons suivant la profondeur de la scène. La texture du papier dynamique est alors projetée sur ce maillage déformé. La figure 30 illustre ce principe.

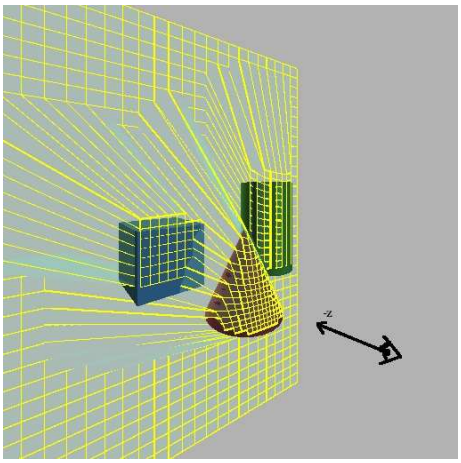

FIG. 30 – Illustration du principe du maillage d´eform´e suivant la texture .

Dans cet exemple, nous créons le maillage à partir d'un point de vue (oeil), puis nous l'observons depuis un autre point de vue afin de mettre en évidence la déformation due à la profondeur.

Pour appliquer cette méthode à la vision stéréo, nous projetons pour chaque "oeil virtuel" un maillage qui lui correspond. Contrairement à la première implantation du papier dynamique, les deux yeux ne voient pas le même motif. Ils voient chacun une projection différente du même objet ce qui devrait permettre au cerveau la reconstruction de l'information de profondeur.

Les résultats obtenus par cette méthode ont été décevants. Nous pouvons remarquer sur le workbench, une légère déformation de profondeur mais l'effet global demeure désagréable.

Toutefois la structure du programme est correcte, et cette première extension de l'algorithme nous a appris beaucoup sur la gestion de "l'effet stéréo". Il sera donc rapidement mis en place une modification de cet algorithme (jouant sur les déformations de textures), qui permettra de résoudre le problème rencontré.

5.3.3 La combinaison lignes de crêtes et rendu Raskar

Même si notre encapsulation RevesAPI dans XP fonctionne correctement, elle pose des problèmes de performance pour les grandes scènes du projet ARCHEOS (de 150 000 à 200 000 polygones). RevesAPI ne dispose pas de toutes les optimisations avancées de Performer (notamment au niveau du culling). Nous avons donc développé deux classes **pfCreaseGeode** et **pfRaskarObject** dans le but d'obtenir un rendu NPR rapide sous XP.

Lignes de crêtes La classe **pfCreaseGeode** est une classe héritant de pfGeode, contenant toutes les lignes de crêtes d'un objet qui lui a été donné en entrée. Cette manière de procéder autorise l'utilisateur à intégrer les lignes de crêtes seules ou en superposition avec l'objet qui leur est attaché. Nous avons bien entendu adapté l'algorithme décrit en 4.2.3 à la structure de données de géométrie de Performer. Toutefois celle-ci est relativement "pauvre" et ne contient pas d'informations de connectivité. Il faut donc la construire. Ce qui complique légerement l'algorithme.

Pour créer un objet "lignes de crêtes" dans un script XP, nous utilisons l'objet *creaseobject* avec comme paramètre :

- *file* : qui est le chemin sur l'objet 3D, dont nous voulons obtenir les lignes de crêtes.
- *angle* : qui est l'angle de seuil nécessaire au calcul des lignes de crêtes.

Rendu Raskar La classe **pfRaskarObject** est assez simple, car l'essentiel du travail est effectué au niveau de la surcharge de la fonction de rendu (*callback* du *draw*). Nous modifions juste les paramètres de couleur de l'objet et le rendons insensible à l'éclairage.

Pour créer un objet "Raskar" dans un script XP, nous utilisons l'objet *raskarobject* avec comme paramètre :

- *file* : qui est le chemin sur l'objet 3D, dont nous voulons obtenir le rendu Raskar.

Interaction des modes Lorsque nous combinons l'utilisation des rendus : standard, crêtes, silhouettes ; il faut prendre garde à le faire dans un ordre convenable, à savoir : objet standard suivi par objet Raskar, puis objet crêtes. Ceci est nécessaire pour que les écritures dans le tampon de profondeur soient correctes, et ainsi que les entités qui doivent être cachées, le soient réellement. C'est en fait le même type de problème que celui posé par les objets transparents.

Pratiquement, nous utilisons les masques de parcours du scenegraphe. Performer met à disposition des méthodes activant ou désactivant des noeuds du scenegraphe, ce qui revient à afficher ou cacher les objets. Nous exploitons ces méthodes en tenant compte de la spécificité du "rendu Raskar".

Algorithme `Dessin global`
Entrée : Scenegraphe S .
Sortie : Rendu de la scene.
Début

```
TravMask(ALL,TRUE) ; // Active le scenegraphe entier.
TravMask(Raskar,FALSE) ; // Cache les objets de" type Raskar".
TravMask(Crease,FALSE) ; // Cache les objets de "type Crêtes".
    Render(S) ; // Traverse le scenegraphe(rendu de la scène "standard")
TravMask(ALL,FALSE) ; // Masque le scenegraphe entier.
TravMask(Crease,FALSE) ; // Cache les objets de "type Crêtes".
TravMask(Raskar,TRUE) ; // Active les objets de" type Raskar".
```

Render_Raskar(S) ; // *Effectue le rendu multi-passes Raskar(rendu de la scène "standard")*
TravMask(Crease,TRUE) ; // *Active les objets de "type Crêtes".*
TravMask(Raskar,FALSE) ; // *Masque les objets de" type Raskar".*
Render(S) ; // *Traverse le scenegraphe(rendu de la scène "standard")*

Fin.

La figure 31 illustre très bien sur l'image de gauche les problèmes posées par un mauvais rendu. Même les crêtes qui devraient être cachées sont visibles, le dessin n'est plus du tout "lisible". L'image de droite est bien meilleure, bien qu'elle présente une certaine saturation des lignes de crêtes inhérentes au modèle.

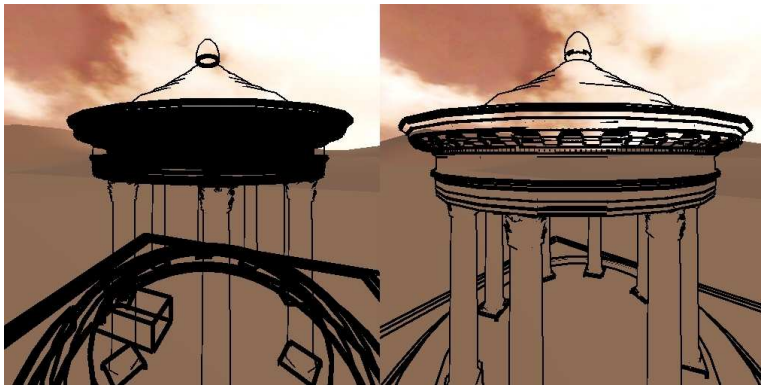

FIG. 31 – Modification des ordres d'affichage.

Les résultats sur le workbench sont tout à fait satisfaisants. Les lignes sont perçues avec leur profondeur.

5.4 La manette de jeux

Les dispositifs d'acquisition des plates-formes de réalité virtuelle sont très onéreux. Leurs prix atteint rapidement plusieurs milliers d'Euros. Le projet a donc entrepris d'utiliser une manette de jeux standard (de trente Euros) comme périphérique d'entrée sur le workbench.

FIG. 32 – La manette de jeux Thrustmaster.

Cette manette de jeux (voir figure 32) comporte dix boutons et deux mini-manettes analogiques, qui permettent un déplacement fluide. Une telle utilisation n'étant bien évidemment pas prévue par les concepteurs des outils de réalité virtuelle, il faut effectuer un certain travail au niveau système. Un ingénieur de l'équipe s'est donc chargé de concevoir le pilote système pour Linux, et l'interface entre ce pilote et trackd (qui "collabore" avec CAVELib).

Nous avons ensuite utilisé ce travail pour intégrer cette manette dans XP, et ainsi pouvoir l'utiliser pleinement sur le workbench. Ceci se fait en modifiant le fichier *xpCAVE.cpp* avec les appels CAVELib

adéquats. Pour le programmeur l'accès à la manette dans XP est alors très simple, et se révèle, à l'usage, très pratique.

Conclusion

Notre travail au sein du projet REVES nous a conduit à mettre au point avec succès un "pont" logiciel entre deux environnements de développement. Nous avons ensuite porté des algorithmes de rendu non-photoréaliste sur une plate-forme immersive avec les outils appropriés.

Ce projet de fin d'étude nous a surtout permis de travailler dans une structure dynamique et stimulante. Nous avons du résoudre des problèmes complexes, travailler avec des algorithmes de haut niveau et améliorer ceux-ci pour nous adapter aux besoins applicatifs. Ainsi nous avons acquis une expérience précieuse en tant qu'utilisateur de Linux et en programmation orientée objet. Nous nous sommes aussi adaptés aux spécificités de l'informatique graphique.

Grâce à nos travaux, il sera donc possible de démontrer l'apport des rendus non-photoréalistes. Et, à la suite de notre projet, l'algorithme de papier dynamique sera amélioré en fonction des résultats que nous avons obtenus. Des résultats pertinents seront donc présentés à la clôture de l'action ARCHEOS en novembre 2003.

Références

[1] M. Barnsley. *Fractals Everywhere.* Academic Press, 1988.

[2] M. Cunzi, J. Thollot, S. Paris, G. Debunne, J.D. Gascuel, and F. Durand. Dynamic canvas for non-photorealistic walkthroughs. In *Graphics Interface*, 2003.

[3] Eric Daniels. Deep canvas in disney's tarzan. In *ACM SIGGRAPH 99 Electronic art and animation catalog*, page 124. ACM Press, 1999.

[4] O. Deussen and T. Strothotte. Computer-generated pen-and-ink illustration of trees. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 13–18. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[5] F. Durand, V. Ostromoukhov, M. Miller, F. Duranleau, and J. Dorsey. Decoupling strokes and high-level attributes for interactive traditional drawing. In *Proceedings of the Eurographics Workshop on Rendering.*, 2001.

[6] R. W. Floyd and L. Steinberg. An adaptative algorithm for spatial grey scale. In *Proceedings of Soc. Inf. Display.*, 1976.

[7] A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 517–526. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[8] T. Isenberg, N. Halper, and T. Strothotte. Stylizing silhouettes at interactive rates : From silhouette edges to silhouette strokes. *Computer Graphics Forum (Proceedings of Eurographics)*, pages 249–258, 2002.

[9] Allison W. Klein, Wilmot W. Li, Michael M. Kazhdan, Wagner T. Correa, Adam Finkelstein, and Thomas A. Funkhouser. Non-photorealistic virtual environments. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 527–534. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[10] Adam Lake, Carl Marshall, Mark Harris, and Marc Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *Proceedings of the first international symposium on Non-photorealistic animation and rendering*, pages 13–20. ACM Press, 2000.

[11] B. Lintermann and O. Deussen. Interactive modeling of plants. *IEEE Computer Graphics and Applications*, 1999.

[12] L. Markosian, M. A. Kowalski, S. J. Trychin, L. D. Bourdev, D. Goldstein, and J. F. Hughes. Real-time nonphotorealistic rendering. *Computer Graphics*, 31(Annual Conference Series) :415–420, 1997.

[13] Barbara J. Meier. Painterly rendering for animation. *Computer Graphics*, 30(Annual Conference Series) :477–484, 1996.

[14] Alex Mohr and Michael Gleicher. Non-invasive, interactive, stylized rendering. In *Symposium on Interactive 3D Graphics*, pages 175–178, 2001.

[15] J. Northrup and L. Markosian. Artistic silhouettes : A hybrid approach, 2000.

[16] R. Raskar. Hardware support for non-photorealistic rendering, 2001.

[17] R. Raskar and M. Cohen. Image precision silhouette edges. In *Symposium on Interactive 3D Graphics*, 1999.

[18] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. Silhouette clipping. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 327–334. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[19] R. Shepars. Circularity in judgements of relative pitch. *J. Acoust.Soc.Am*, 1964.

Annexes

A pseudo-code papier dynamique

Algorithme `Composition_texture`
Entrée : Texture T, nombre d'octave n, facteur de zoom z.
Sortie : Texture T' de papier dynamique .
Initialisation
AdjustZoom(z) ;// *Ramène le coefficient de zoom z dans [1.0,2.0[.*
$\omega \leftarrow 1.0$;// *Fréquence spatiale.*
$\delta \leftarrow 2^n/(2^n - 1)$;// *Facteur de normalisation tel que* $\delta * (1 + 1/2 + ... + 1/2^n) = 1.0$.
Début
ClearScreen() ; // *Efface les buffers.*
SaveViewport(vp[]) ; // *Sauve les paramètres écran.*
PushMatrix(ALL) ; // *Empile les matrices de transformation, projection, texture.*
LoadIdentity(ALL) ; // *Définit ces matrices à l'identité.*
SetOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0) ; // *Définit la projection courante comme orthographique.*
for i=0 **to** n-1 **do**
// *Effectue le fondu entre les octaves en jouant sur le coefficient de transparence* α.
if i=0 **then**
$\alpha \leftarrow (z - 1.0)$; // *Pour la* 1^{er} *octave.*
else
if i=(n-1) **then**
$\alpha \leftarrow (2.0 - z)$; *Pour la* n^{eme} *octave.*
else
$\alpha \leftarrow (1.0)$; *Pour les autres octaves.*
fi ;
fi ;
BlendColor($\frac{\alpha * \delta}{z}$) ; // *Définit le coefficient de transparence de la* i^{eme} *octave.*
BeginGeometry(QUAD) ; // *Définit la géométrie pour projeter la texture.*
SetUV(0, 0) ; SetVertex(0.0f, 0.0f, 0.0f) ;
SetUV(w, 0) ; SetVertex(1.0f, 0.0f, 0.0f) ;
SetUV(w,w) ; SetVertex(1.0f, 1.0f, 0.0f) ;
SetUV(0, w) ; SetVertex(0.0f, 1.0f, 0.0f) ;
EndGeometry() ;
$\omega \leftarrow 2.0 * \omega$;// *Double la fréquence saptiale.*
$\delta \leftarrow 0.5 * \delta$;// *Atténue les hautes fréquences.*
od ;
CopyImage(T') ; // *Copie le buffer image courant(texture procédural ;) dans la texture T'.*
PopMatrix(ALL) ; // *Restaure les matrices de transformation, projection, texture.*
LoadViewport(vp[]) ; // *Restaure les paramètres écran.*
renvoie(T') ;
Fin

Algorithme `Affichage_texture`
Entrée : Texture dynamique : T, position : P, ancienne position :oldP, activation du spherical warp : warp, camera C.
Sortie : Affichage du papier dynamique .
Initialisation

Début
PushMatrix(ALL) ; // *Empile les matrices de transformation, projection, texture.*
LoadIdentity(ALL) ; // *Définit ces matrices à l'identité.*
ComputeDisplace(P,oldP) ; // *Calcul les déplacements à donner à la texture(via matrice de texture).*
if warp **then**// *Activation du spherical Warp.*
PrintTextureWarp(T,C) ; // *Affiche la texture en appliquant le spherical warp.*
else
PrintTexture(T) ; // *Affiche la texture.*
fi ;
PopMatrix(ALL) ; // *Restaure les matrices de transformation, projection, texture.*
Fin

Algorithme `Rendu_Papier`
Entrée : Texture : T, Scene : S.
Sortie : Rendu du papier dynamique .
Initialisation
Début
T'← Composition_texture(T) ; // *Compose la texture.*
Rendu(S) ; // *Affiche la scène.*
Affichage_texture(T') ;// *Affiche la texture a l'écran.*
Fin

B pseudo-code lignes de crêtes

Le pseudo-code suivant nécessite pour s'appliquer fidèlement de disposer d'une structure de données évoluées, qui contient les informations de connectivité des modèles. C'est le cas avec les Indexed Face Set (IFS) de RevesAPI, mais pas avec la structure de Performer.
Algorithme `Crêtes`
Entrée : Noeud de géométrie G, seuil d'angle α_0.
Sortie : Noeud de géométrie Crease.
Initialisation
$n \leftarrow G.GetNumberEdge()$;// *Sauve le nombre d'arêtes de G.*
Début
for i=0 **to** n-1 **do**
$edge \leftarrow G.GetEdge(i)$; // *Récupère la i^{eme} arête.*
if edge.GetNumberFaces()==1 **then** // *Test si edge n'est liée qu'à un polygone.*
Crease.AddEdge(edge) ; //*C'est une arête de bord, nous l'ajoutons à Crease.*
fi ;
if edge.GetNumberFaces()==2 **then** // *Test si edge est liée à 2 polygones.*
$face0 \leftarrow edge.GetFace(0)$; //*Récupère la 1^{ere} face.*
$face1 \leftarrow edge.GetFace(1)$; //*Récupère la 2^{eme} face.*
// *Test si une normale est définie pour chaque face (et non par sommets).*
if face0.HasFaceNormal() **and** face1.HasFaceNormal() **then**
$\vec{vec}0 \leftarrow edge.GetFace(0)$; //*Récupère la normale à la 1^{ere} face.*
$\vec{vec}1 \leftarrow edge.GetFace(1)$; //*Récupère la normale à la 2^{eme} face.*
if $abs(\vec{vec}0.\vec{vec}1) < cos(\alpha_0)$ **then** //*Test de l'angle entre les faces.*
Crease.AddEdge(edge) ; //*C'est une crête, nous l'ajoutons à Crease.*
fi ;

```
        fi ;
   od
renvoie(Crease);
Fin.
```

C Classe pfrvGroup

```
class pfrvGroup : public pfGroup {
   private :
      //Declaration RevesAPI
      CsgMaterialBuilder* m_MtlBuilder; // Material builder
      CsgIFSBuilder * m_IFSBuilder;// Geometry builder
      CsgNode* rv_rootnode;//Noeud initial du graph RevesAPI
      CrvScene *rvScene;
      CtemplRenderer* m_Renderer;
      float CreaseAngle;
      float minCrease;
      //Declaration pf
      static pfType* classType_;
   public :
      //Typage et surcharge pf :
      static void init(void);
      static pfType* getClassType(void);
      static int preDraw( pfTraverser *trav, void *data);
     //Construction
      pfrvGroup();
      pfrvGroup(char* filename);
       pfrvGroup();
      //Conversion :
      pfNode * rvWrap(pfGroup *pgroup);
      CsgNode* Convert(pfNode * node);
      CsgTransform * ConvertTransform(pfSCS *SCS);
     CsgTransform * ConvertTransform(pfDCS *DCS);
      CsgNode* ConvertGeometry(pfGeode *Geode);
     CsgNode* ConvertGeometry(pfGeoSet *GeoSet);
      CsgMaterial * ConvertMaterial(pfGeoState *state);
      //Affichage des scenegraphes locaux :
      void ShowTree(pfNode * node,int rvlevel=0);
      void ShowrvTree(CsgNode *node=NULL,int rvlevel=0);
      //Retour information :
      CsgNode* GetrvRoot(void);
      CrvScene* GetrvScene(void);
      CtemplRenderer* Getrenderer(void);
};
```

D Utilisation de CrdDisplayListCrease

Il est assez simple de re-utiliser les lignes de crêtes dans un module de rendu RevesAPI. Il suffit après avoir obtenu un pointeur sur le constructeur de géométrie (*GeometryBuilder*) avec la commande :

```
m_pDLBuilder = static_cast <CrdDisplayListBuilder*>
 (l_pTheGeoManager->GetGeometryBuilder(l_iDisplayListBuilderId)) ;
```

d'indiquer que nous voulons utiliser la classe **CrdDisplayListCrease** ce qui se fait avec :

```
m_pDLBuilder->SetDisplayListInstance(new CrdDisplayListCrease());
m_pDLBuilder->InvalidateAllLists();
```