UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

MASTER HANDI
Domaine : Sciences, Technologies, Santé (STS)
Mention : Ingénierie et Cognitions
Spécialité : Technologies et Handicap
Rapport de stage de Master 2

Développement d'une application d'exploration d'objets 3D numérisés par bras à retour d'effort

Johana Bodard

Directeur de stage : M. Gérard Uzan
Lieu de stage : Laboratoire THIM, Université Paris 8

Coordinateur : M. Jaime López Krahe

Saint Denis, septembre 2013

Laboratoire THIM

LABORATOIRE D'EXCELLENCE DES ARTS ET MÉDIATIONS HUMAINES

Remerciements

Je tiens à remercier tous ceux qui m'ont permis de faire ce stage et en particulier M. Gérard Uzan, mon tuteur de stage pour son aide et ses conseils, et M. Jaime López Krahe.

Je remercie également Benoit, Tahar et Godefroy et tous ceux qui m'ont accompagnée durant cette année au sein du master HANDI.

Enfin, je tiens à remercier mon conjoint qui m'a soutenue dans ma reprise d'études.

Résumé

Les Archives Nationales, en partenariat avec le laboratoire THIM de Paris 8, ont initié en 2012, un projet dont le but est de rendre accessible au plus grand nombre et notamment à un public malvoyant, le patrimoine médiéval français. En particulier, les Archives Nationales souhaitent étudier les possibilités offertes par les bras à retour d'effort pour une exploration haptique de sceaux médiévaux qui ont été numérisés.

Le but du travail réalisé pendant ce stage est de développer une application permettant d'explorer des sceaux par l'intermédiaire d'un dispositif haptique. Pour que cette exploration soit efficace, il faut appliquer aux sceaux des transformations qui permettent de saisir plus facilement leur forme et leurs reliefs. Ces transformations font appel à des techniques de traitement d'images et de manipulation de maillage 3D.

Table des matières

Index des illustrations

1 Introduction et contexte

1.1 Introduction

Les Archives Nationales conservent une importante collections de sceaux médiévaux qu'ils souhaitent rendre accessibles sous forme numérisée à tous et en particulier aux personnes aveugles et malvoyantes. Quand on est déficient visuel, le meilleur moyen pour percevoir des objets est de les explorer par le toucher. Pour explorer des objets numérisés, l'utilisation de bras à retour d'effort permet d'exploiter la modalité haptique. Un projet du LabEx ART-H2H qui regroupe les Archives Nationales et le laboratoire THIM de l'Université Paris 8, vise à développer une application permettant d'explorer ces sceaux par l'intermédiaire d'un dispositif haptique. Les travaux portent sur les transformations dynamiques et statiques que l'on peut appliquer à un objet 3D pour améliorer sa perception, sur la mise en place d'une exploration guidée et commentée des sceaux et sur les possibilités d'exploration à distance.

C'est dans ce contexte que s'est déroulé mon stage au sein du laboratoire THIM. L'objectif étant de fournir une application d'exploration par bras à retour d'efforts d'objets 3D numérisés, mon travail a d'abord consister à développer l'interface graphique de cette application, puis à étudier et développer les transformations : simplification, changement d'échelle, suppression de motifs ou de plans, déformations, etc. Le travail concernant le guidage et les déplacements du bras haptique est réalisé par Godefroy Clair, développeur au sein du laboratoire. L'interface graphique doit notamment permettre d'ajouter des commentaires audio pour l'exploration guidée des sceaux.

Ce rapport présente le travail que j'ai réalisé pendant ce stage. Dans les paragraphes suivants, je présente l'environnement de travail dans lequel s'est déroulé mon stage. La seconde partie porte sur les recherches que j'ai effectuées, préalablement au développement, relatives à la perception haptique, la simplification et la segmentation d'objets 3D. La troisième partie concerne le développement informatique de l'application. Enfin, dans la quatrième partie, je conclus sur le travail effectué, sur ce qu'il reste à faire et sur les difficultés que j'ai pu rencontrer.

1.2 Laboratoire THIM et LabEx ARTS-H2H

Ce stage s'est déroulé au sein du laboratoire THIM - CHART EA 4004

(Technologies, Handicaps, Interfaces et Multimodalités - Cognitions Humaine et ARTificielle) de l'Université Paris 8. Le laboratoire THIM travaille sur des projets pluridisciplinaires (traitement d'image, informatique, ergonomie, etc.) dans le but de concevoir des aides techniques ou d'améliorer l'accessibilité des informations pour les personnes en situation de handicap.

Le travail que j'ai effectué a été encadré par M. Gérard Uzan, ingénieur de recherche et ergonome, au sein de ce laboratoire.

Ce stage s'intègre dans un projet plus vaste soutenu par le LabEx ARTS-H2H, laboratoire d'excellence des arts et médiations humaines, qui regroupe différentes institutions et établissements d'enseignement supérieur, dont les Archives Nationales et l'université Paris 8, autour de thématiques de recherches communes mêlant à la fois les arts, les sciences cognitives, les médiations humaines et les nouvelles technologies.

1.3 Ressources à disposition

1.3.1 Sceaux

Un sceau est une empreinte faite sur une matière malléable telle que la cire à partir d'une matrice en relief qui représente un roi, une ville, un métier, etc. Il permet d'authentifier un document et de garantir sa confidentialité.

Quatre sceaux médiévaux ont été sélectionnés pour effectuer les tests (cf. illustrations 1 à 5) :

- un sceau de majesté représentant Philippe Auguste ;
- un sceau équestre représentant Robert de Dreux ;
- un sceau de dame en forme d'ogive représentant Jeanne de Châtillon ;
- un sceau à deux empreintes de la ville de Marseille.

Illustration 1: Sceau en plâtre de Philippe Auguste

Illustration 2: Sceau en plâtre de Robert de Dreux

Illustration 3: Sceau numérisé de Jeanne de Châtillon

Illustration 4: Sceau numérisé de la ville de Marseille - recto

Illustration 5: Sceau numérisé de la ville de Marseille - verso

1.3.2 Bras à retour d'effort

Le bras que j'ai utilisé pour effectuer mes tests est un Phantom Omni de Sensable (cf. illustration). Ces caractéristiques techniques sont les suivantes :

- un seul point de contact ;

- six degrés de liberté en détection (position dans l'espace suivant les axes x, y et z, et orientation suivant les axes tangage, roulis et lacet) ;

- retour de force sur les axes x, y, z ;

- reproduit les mouvements de la main pivotant au niveau du poignet ;

- connexion en FireWire.

Au cours de mon stage, le laboratoire a acquis un autre modèle de bras, le Phantom Premium 1.5 6/DOF de Sensable, qui fournit le retour de force sur les six axes de liberté et qui permet de changer l'actionneur d'extrémité. Il reproduit les mouvements de l'avant-bras pivotant au niveau du coude. Nous n'avons pas pu faire fonctionner ce bras pendant la durée de mon stage et je ne l'ai donc pas utilisé.

Ces bras sont fournis avec le kit de développement OpenHaptics [OH], un ensemble d'outils pour développer des applications haptiques.

Illustration 6: Bras Phantom Omni

2 État de l'art

2.1 Perception d'un objet

Avant de commencer à développer une application haptique, il est important de comprendre comment cette perception fonctionne, notamment lors de l'utilisation de dispositifs haptiques tels un bras à retour d'efforts.

E. Gentaz et Y. Hatwell identifient deux types de perception manuelle [1] :

- la perception tactile, passive, lorsqu'on déplace un objet sur une partie de la main immobile ;

- la perception haptique, active, lorsque la main se déplace pour explorer un objet.

Dans le cadre de ce projet, c'est la main par l'intermédiaire d'un dispositif haptique qui est active et qui se déplace pour explorer l'objet. On s'intéressera donc au deuxième type de perception.

2.1.1 Exploration haptique d'objets réels

La perception haptique fait appel au sens du toucher, ou tact, par l'intermédiaire de plusieurs mécano-récepteurs présents dans la peau, à la proprioception qui est la connaissance de l'état de son propre corps et à la kinesthésie qui est la sensation de la position relative des différentes parties du corps et des mouvements et forces qui leurs sont appliquées. La kinesthésie fait appel à de nombreux mécano-récepteurs proprioceptifs situés au niveau des articulations, des tendons et des muscles [2].

S. Lederman et R. Klatzky ont montré que l'exploration haptique d'un objet pour en extraire des propriétés matérielles ou spatiales utilise des procédures exploratoires [3] : le frottement latéral, la pression, le contact statique, le soulèvement, l'enveloppement, le suivi de contours. Pour chaque propriété (texture, forme globale, forme exacte, etc.), elles ont identifié les procédures nécessaires, suffisantes et optimales, ainsi que leur efficacité. Les résultats montrent que si la perception haptique est très efficace pour déterminer la texture d'un objet via la procédure de frottement latéral, elle est moins performante pour en déterminer les propriétés spatiales. La forme globale peut être obtenue de façon optimale avec la procédure d'enveloppement et de façon suffisante avec les procédures de suivi de contours, de contact statique et de

soulèvement. Mais il n'existe aucune procédure exploratoire haptique optimale pour connaitre la forme exacte d'un objet : la seule procédure applicable est le suivi de contours. De plus, c'est une procédure lente.

Ces mêmes auteurs ont montré que l'exploration d'un objet se déroulait en deux temps [4]. La première étape utilise essentiellement deux procédures exploratoires rapides et non spécifiques à une propriété, l'enveloppement et le soulèvement, pour obtenir une représentation globale de l'objet. La deuxième étape utilise des procédures exploratoires plus précises, comme le suivi de contours, pour extraire la propriété souhaitée.

2.1.2 Exploration d'objets 3D virtuels par l'intermédiaire d'un dispositif haptique

Dans ce projet, l'exploration est réalisée sur des objets 3D virtuels par l'intermédiaire d'un périphérique haptique qui possède un seul point de contact avec la surface à explorer. Quand on utilise ce type de dispositif mono-point, le nombre d'informations disponibles est fortement réduit : l'enveloppement n'est plus possible et on perd les informations cutanées. Dans ce cas, seul le suivi de contours nous permet d'obtenir les deux propriétés des objets qui nous intéresse le plus : la forme globale et la forme exacte.

Le potentiel des périphériques haptiques mono-point comme source unique d'information a été étudié dans de nombreuses expériences. Dans [5], il a été démontré que les textures virtuelles sont perçues aussi efficacement que les textures réelles, mais que la perception des formes des objets virtuels n'est pas aussi bonne que celle des objets réels. Elle est sujette à plus d'erreurs et plus lente, en particulier quand les objets sont petits [6]. L'efficacité diminue aussi avec la complexité de l'objet [7]. De plus, les objets sans arêtes ou côtés comme les sphères sont très difficiles à percevoir avec le suivi de contours [7]. Dans [8], les auteurs proposent deux méthodes pour améliorer la perception haptique d'objets 3D virtuels. La première consiste à entraîner la personne aveugle à percevoir un objet avec un périphérique haptique. En effet, avec seulement quelques heures d'entraînement, il est possible d'améliorer sensiblement l'efficacité de l'exploration chez la plupart des personnes. Cependant, une minorité des participants n'arrive pas à améliorer ses performances après un entrainement. La seconde méthode consiste à adapter le périphérique haptique en augmentant le nombre d'informations transmises lors d'un contact. Les auteurs suggèrent d'ajouter des points de contact : le nombre d'erreurs est réduit d'un facteur 2,5 quand on utilise deux doigts au lieu d'un seul pour explorer un objet réel, mais il ne diminue pratiquement plus quand on utilise trois doigts ou plus. Ils proposent également d'augmenter la

quantité d'information à la surface de la peau. S. Lederman et R. Klatzky ont en effet démontré que supprimer les forces qui s'exercent directement à la surface de la peau altèrent les performances notamment dans la perception des propriétés spatiales [9].

Dans [10], les auteurs réalisent une expérimentation avec des objets virtuels géométriques simples, des sphères et des cubes, visualisés de l'intérieur et de l'extérieur à l'aide d'un périphérique haptique mono-point. Les auteurs observent des erreurs dans l'estimation de la taille des objets plus importantes sur les petits objets. La taille des objets perçus de l'intérieur a tendance à être surestimée d'un quart environ. La taille des objets perçus de l'extérieur a tendance à être légèrement sous-estimée. Enfin, les angles sont mal perçus. Dans une autre expérience, ils présentent aux participants des objets complexes comme des canapés ou des chaises, en informant préalablement les participants de ce qu'ils vont percevoir (une étude pilote ayant démontré que les participants ne sont pas capables d'identifier l'objet autrement). Les résultats montrent que les participants sont capables de ressentir les différentes parties de l'objet comme les bras ou les pieds, mais pas de reconstituer l'objet en entier. Bien que ces objets complexes soient composés d'éléments simples comme des cubes, il est difficile pour eux d'appréhender la forme globale de l'objet. En effet, la sonde du périphérique se perd facilement dans l'espace haptique. Par exemple, quand un participant parcourt un côté d'un cube, arrivé à la fin, la sonde glisse dans l'espace haptique et le participant doit retrouver l'objet. Le participant perd la trace de ce qu'il était en train d'explorer parce qu'il n'y a pas de point de référence. Les résultats montrent également que les participants n'ont pas tous la même représentation mentale de l'espace virtuel (certains le placent à l'intérieur du périphérique, d'autres à l'extérieur) et de la partie de la sonde en contact avec l'objet.

2.1.3 Principes de conception pour une meilleure perception

Ce travail de recherche préliminaire montre que l'exploration haptique d'un objet virtuel pour en connaitre la forme est moins performante que l'exploration d'un objet réel avec les mains, car elle se limite souvent au suivi de contours de l'objet à cause des limitations techniques imposées par le périphérique. De plus, la sonde du périphérique peut se perdre facilement dans l'espace haptique si l'utilisateur n'est pas guidé. Il apparait également nécessaire de simplifier les objets complexes, d'accentuer les arêtes et contours et d'apporter un maximum d'informations sur l'objet lors de l'exploration.

2.2 Représentation numérique d'objets 3D

2.2.1 Maillage

Pour pouvoir être manipulés sur un ordinateur, les objets 3D numérisés doivent être représentés sous une forme géométrique discrète adaptée aux capacités de calcul informatique. Le maillage de surface est très utilisé pour représenter un objet en modélisant uniquement l'enveloppe qui l'entoure. Un maillage est constitué d'un ensemble de points, les sommets, reliés entre eux par des arêtes formant ainsi des faces. L'un des types de maillage les plus utilisés est le maillage triangulaire, la triangulation de Delaunay d'un ensemble de points du plan permet d'obtenir un tel maillage. Mais il est également possible d'utiliser des maillages dont les faces sont des carrés, des pentagones, etc.

Il existe plusieurs structures de données informatiques permettant de représenter des maillages. Les paragraphes qui suivent décrivent les structures les plus couramment utilisées ainsi que leurs avantages et leurs inconvénients.

2.2.2 Structures de données orientées sommet

Ces structures de données sont faciles à implémenter et utilisent peu d'espace mémoire puisqu'elles ne stockent que des références vers des sommets. Mais elles ne contiennent aucune référence explicite vers les arêtes et les faces du maillage. Il faut donc à chaque fois parcourir l'ensemble de la liste des sommets pour les reconstruire. De même, les opérations sur les arêtes et les faces ne sont pas aisées.

Il existe deux structures de données orientées sommet :

Vertex-Vertex : liste des sommets, chaque sommet stocke la liste de ses sommets voisins.

Corner-table : liste des sommets construite de telle manière que quand on la parcourt on définit implicitement les faces.

2.2.3 Structures de données orientées faces

Cette structure de données utilise peu d'espace mémoire et est performante pour le rendu graphique. Mais il est difficile de modifier le maillage. Les opérations sur les arêtes (qu'il faut reconstruire) et les faces ne sont pas aisées.

Il n'existe qu'une structure de données orientée face :

Face-vertex ou ***Indexed Face Set*** :liste des sommets et liste des faces, chaque face stocke la liste des indices de ces sommets dans la précédente liste.

2.2.4 Structure de données orientées arêtes

Ces structures de données permettent de parcourir et de modifier facilement un maillage, mais elles stockent beaucoup plus de références que les structures précédentes et utilisent donc beaucoup plus d'espace mémoire.

Il existe trois structures de données orientées arêtes :

Winged-edge [11] :chaque arête possède des pointeurs vers les deux sommets adjacents, les deux faces adjacentes et les quatre arêtes qui partagent les mêmes sommets et faces

Half-edge [12] :chaque arête est séparée en deux demi-arêtes opposées. Chaque demi-arête possède des pointeurs vers le sommet destination, la demi-arête opposée, la face adjacente, la demi-arête suivante dans la face.

Quad-edge [13] :chaque arête e est séparée en quatre arêtes e1, e2, e3 et e4. Cette structure de données permet de représenter en même temps le maillage et son dual. Les arêtes e1 et e2 sont les deux demi-arêtes opposées de l'arête e. Les arêtes e2 et e4 sont les deux demi-arêtes opposées de la duale de e.

2.3 Simplification de maillage

Les algorithmes de simplification de maillage ont été développés à partir des années 90 pour optimiser le traitement informatique de maillages pouvant contenir plusieurs milliers de faces. Ils tentent de réduire le nombre de faces tout en préservant la forme de l'objet. Ils sont utilisés pour simplifier des modèles numérisés à de très hautes résolutions (plusieurs centaines de milliers voir plusieurs millions de faces), réduire le temps de transfert d'objets 3D dans un réseau ou créer des modélisations multi-résolution d'un objet 3D (plus un objet est loin plus le niveau de détail est faible). L'intérêt d'utiliser un algorithme de simplification dans le cadre de ce projet est avant tout de faciliter l'exploration haptique des sceaux.

Il existe deux types d'algorithmes pour simplifier un maillage :

1. Les algorithmes de rassemblement de sommets (*vertex clustering*) [14] : ils consistent à placer un maillage dans une boite que l'on divise en grilles. Dans chaque cellule de la grille, les sommets sont rassemblés en un seul sommet. Ce type d'algorithmes autorise d'importantes modifications de la topologie du maillage. Les résultats obtenus sont souvent médiocres, même s'il est possible d'améliorer la qualité de la simplification en réduisant la taille des cellules.
2. Les algorithmes qui appliquent itérativement des opérations de simplifications locales : ils ne modifient qu'une petite partie du maillage à chaque itération. L'opération appliquée au maillage peut être la suppression d'un sommet [15], l'effondrement d'une arête [16] ou la contraction d'une paire de sommets [17].

Ce dernier type d'algorithmes suit les étapes suivantes :

1. On choisit une condition d’arrêt : le nombre de face minimum par exemple.
2. On détermine toutes les opérations valides et on leur assigne un coût qui représente la quantité de changement appliqué au maillage.
3. On choisit l’opération avec le coût le plus faible et on l’applique. Dans le cas de la suppression d'un sommet, on supprime le sommet et ses faces adjacentes puis on triangule le trou obtenu. Dans le cas de l'effondrement d'une arête, on fusionne les deux sommets d'une arête en un nouveau sommet ce qui supprime les deux faces adjacentes à cette arête. La contraction de paire de sommets permet de fusionner deux sommets qui ne sont pas adjacents à la même arête en un nouveau sommet si la distance qui les séparent est inférieure à un seuil prédéfini. Dans ces deux derniers cas, on peut utiliser une fonction de placement pour déterminer la position du nouveau sommet qui minimise les modifications apportées au maillage. L'opération le plus utilisée est l'effondrement d'arête, car c'est la plus simple à implémenter.
4. On recalcule le coût des opérations valides pour la partie du maillage modifiée.
5. On retourne à l'étape 3. tant que la condition d’arrêt en 1. n’est pas atteinte.

La qualité de la simplification dépend de la fonction choisie pour calculer le coût d'une opération. Mais une fonction de coût trop complexe conduit à des temps d'exécution trop longs. Dans [18], les auteurs comparent plusieurs fonctions de coût en évaluant la qualité de la simplification et le temps d'exécution. L'algorithme de M. Garland et P. S. Heckbert [17] obtient les meilleurs résultats. Mais il consomme beaucoup d'espace mémoire car il

stocke, pour chaque sommet, une matrice 4x4 qui est utilisée pour calculer le coût d'une contraction. P. Lindstrom et G. Turk ont proposé un algorithme [19] basé sur l'effondrement d'arêtes et qui ne nécessite pas de conserver des informations supplémentaires. Dans [20], ils montrent que leur algorithme produit en moyenne moins d'erreurs que l'algorithme de M. Garland et P. S. Heckbert tout en nécessitant beaucoup moins d'espace mémoire.

Pour simplifier l'exploration haptique d'objets 3D, il est important de simplifier ces objets mais il est également important de conserver les arêtes et les creux car ils permettent d'extraire des contours, des formes dans l'objet. Or les algorithmes que j'ai présenté ne prennent pas en compte ces caractéristiques des objets et ne permettent donc pas de les préserver efficacement dans le cas d'une simplification à un niveau de détail très bas. C'est ce qui nous amène au dernier paragraphe de cette partie qui concerne la segmentation d'image.

2.4 Segmentation d'images 3D

La segmentation d'images est une opération qui permet de séparer une image en plusieurs régions suivant des critères prédéfinis (par exemple la couleur ou l'intensité lumineuse d'un pixel). Elle permet notamment de détecter les contours d'une image. Un exemple de technique de segmentation est la ligne de partage des eaux [21] qui est appliquée sur une image en niveaux de gris. L'image est considérée comme un relief topographique, le niveau de gris de chaque pixel correspondant à son altitude. Après avoir simulé l'inondation du relief, on obtient plusieurs bassins versants limités par la ligne de partage des eaux. Ces bassins versants correspondent aux régions de l'image.

La ligne de partage des eaux est une des applications de la morphologie mathématique, une théorie mathématique utilisée en traitement d'images 2D et 3D. Elle consiste à étudier les relations entre un ensemble, appelé élément structurant, et les données d'une image. L'élément structurant est appliqué en tout point de l'image au moyen de quatre opérateurs de base :

- la dilatation binaire qui utilise la somme de Minkowski. La somme de Minkowski de deux ensembles de points P et Q dans $\mathbb{R}^d$, notée $P \oplus Q$, est l'ensemble $\{p+q, p \in P, q \in Q\}$;

- l'érosion binaire qui utilise la soustraction de Minkowski. La soustraction de Minkowski de deux ensembles de points P et Q dans $\mathbb{R}^d$, notée $P \ominus Q$, est l'ensemble $\{p+q, p \in P, q \in Q\}$;

- l'ouverture binaire qui est une érosion suivie d'une dilatation ;

- la fermeture binaire qui est une dilatation suivie d'une érosion.

A. P. Mangan et R. T. Whitaker utilisent la ligne de partage des eaux pour segmenter un maillage 3D [22]. Ils calculent la hauteur d'un sommet en fonction de la courbure du maillage. D'autres algorithmes de segmentation de maillage 3D qui ne se basent pas sur la morphologie mathématique ont été développés [23]. Mais par manque de temps et parce que je n'ai pas toutes les connaissances requises pour bien appréhender ces algorithmes, je n'ai pas pu approfondir autant que je le voulais mes recherches sur ce sujet.

3 Réalisation

3.1 Présentation générale

L'application a été développée en C++ avec l'environnement de développement intégré Eclipse/CDT 4.2 sous Windows XP et Debian Linux. Pour disposer du même environnement sous Windows et sous Linux, j'ai utilisé les outils de développement et de compilation de MinGW 4.7.2 sous Windows.

L'application utilise plusieurs bibliothèques logicielles C ou C++ :

- **wx-widgets 2.8.12** [WX]: une bibliothèque graphique C++ utilisée pour développer l'interface graphique ;

- **OpenGL 2.1** et **GLU 1.3** : OpenGL est une interface de programmation C pour le rendu graphique en 2D et 3D, GLU apporte des fonctionnalités de plus haut niveau à OpenGL. Ces deux bibliothèques sont utilisées pour le rendu graphique des sceaux en 3D. Elles sont déjà installées sur la plupart des ordinateurs ;

- **OpenHaptics 3.10.5** [OH]: un ensemble d'outils en C et C++ pour la gestion des bras Sensable Phantom Omni et le développement d'applications haptiques. Il est utilisé pour l'interface haptique et le rendu haptique des sceaux ;

- **CGAL 4.2** [CGAL] : une bibliothèque de calcul géométrique en C++. Elle fournit la structure des données et les algorithmes qui permettent de manipuler les sceaux dans l'application ;

- **Boost 1.53.0** [BOOST] : une bibliothèque C++ qui apporte de nouvelles fonctionnalités à la bibliothèque C++ standard et qui est également utilisée par CGAL.

Tous ces logiciels sont multiplate-formes (ils disposent au moins de versions pour les systèmes d'exploitation Windows, Linux et Mac OS X) et, à l'exception d'OpenHaptics, ils sont tous *open source*. La version d'OpenHaptics utilisée est l'Academic Edition qui est gratuite pour un usage non commercial.

Pour simplifier la programmation de l'application, j'ai séparé le développement en deux logiciels :

- **une bibliothèque logicielle de manipulation de maillages 3D** qui dépend uniquement de CGAL et de Boost ;

- **l'application d'exploration des sceaux**, qui comprend l'interface graphique et haptique et qui dépend de la précédente bibliothèque, de wx-

widgets, d'OpenGL/GLU, d'OpenHaptics et de Boost.

La bibliothèque de manipulation de maillage implémente une **classe Mesh** qui permet de charger un fichier de description de modèle 3D dans un maillage triangulaire et d'appliquer à ce maillage diverses opérations (mise à l'échelle, simplification, etc.).

L'application implémente plusieurs classes :

- la **classe Sceau** permet de charger un sceau, de l'afficher et de lui appliquer diverses opérations (zoom, simplification, etc.). Elle utilise la classe Mesh pour stocker et manipuler la géométrie des sceaux ;

- la **classe Device** gère le périphérique haptique ;

- **diverses classes qui héritent de classes wx-widgets** pour l'interface graphique.

Le code des classes Mesh, Sceau et Device sont présentés en annexe.

3.2 Interface graphique

3.2.1 wx-widgets

L'interface graphique a été programmée dans un premier temps avec la MFC (Microsoft Foundation Class) de Windows, un ensemble de classes en C++ qui permet de développer des applications graphiques compatibles avec l'interface de programmation Win32. Afin de rendre l'application accessible au plus grand nombre, l'interface a été reprogrammés avec la bibliothèque C++ wx-widgets. wx-widgets permet de développer des applications pour Windows, Linux et Mac OS tout en conservant l'apparence native de chaque environnement. Elle est assez similaire à la MFC de Windows dans son fonctionnement.

Classe MyApp

C'est la classe qui permet le lancement de l'application. Elle crée une fenêtre avec la classe MyFrame, le canevas OpenGL dans lequel on effectue le rendu visuel, un panneau qui permet d'afficher des informations à l'utilisateur et initialise le périphérique haptique.

Elle possède une méthode *OnIdle* qui permet le rafraichissement de l'affichage OpenGL toutes les 50 ms quand l'application n'est pas occupée.

Classe MyFrame

Elle permet de définir une fenêtre dans laquelle on place les menus, la barre d'outils, qui reprend les commandes du menu les plus utilisées, le canevas OpenGL, le panneau ainsi qu'une barre de statut (cf. illustrations 7 et 8).

Les différents menus sont :

- le **menu Sceaux** permet de charger un sceau au format OBJ ou OFF, d'importer un sceau, c'est-à-dire de le placer dans une bibliothèque de sceaux avec ses commentaires et le document qu'il authentifie, d'exporter un sceau au format OBJ, de l'imprimer, de configurer l'application et de la quitter ;

- le **menu Exploration** permet d'appliquer différents zooms au sceau, de sélectionner un plan, un élément (personnage, fleur de lys, etc.), de déformer le sceau statiquement ou dynamiquement et de renforcer ses reliefs ;

- le **menu Guidage** permet d'amener le bras haptique à un élément et de lire les commentaires associés ;

- le **menu Commentaires** permet de créer, modifier ou supprimer des commentaires et de les associer à un sceau.

Actuellement, seule la commande permettant de charger un sceau et les commandes de zoom et de simplification dans le menu Exploration sont implémentées.

Classe MyGLCanvas

Elle gère le canevas dans lequel on dessine les sceaux et le curseur du bras haptique.

Classe MyPanel

Elle gère le panneau qui est actuellement utilisé pour afficher des informations sur le sceau ou le périphérique haptique, comme le nombre de sommets du maillage ou la position du curseur du bras, pour faciliter le débogage de l'application. Mais elle pourra être utilisée pour afficher les commentaires associés aux sceaux quand ceux-ci seront intégrés à l'application.

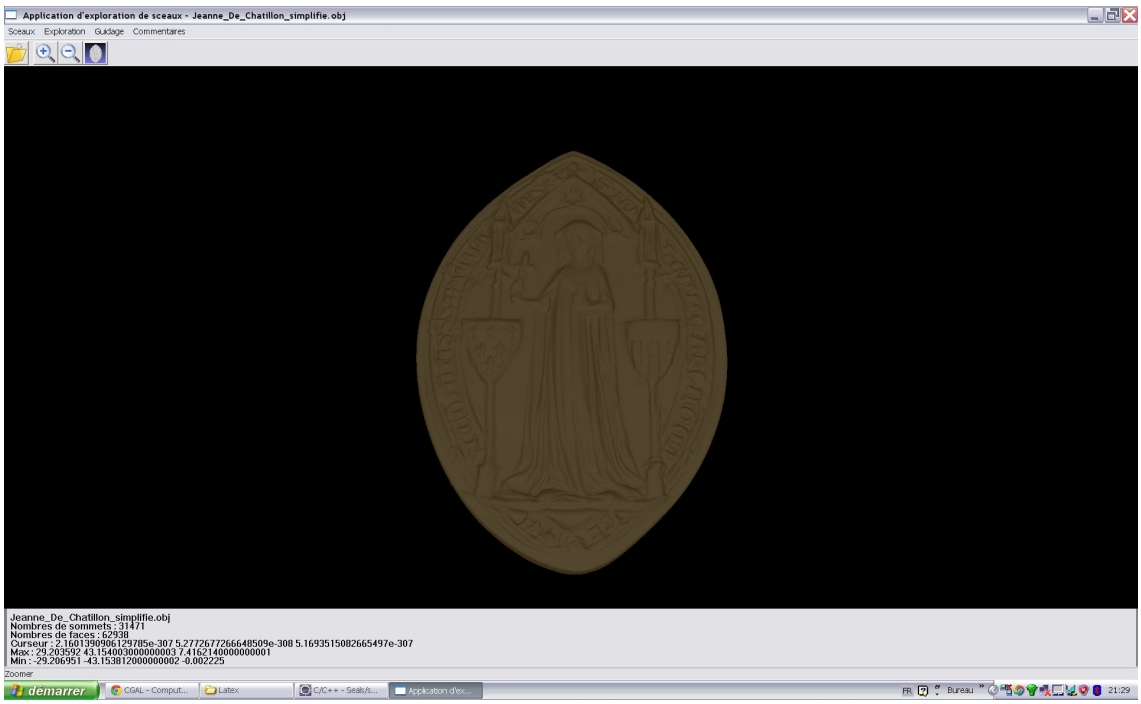

Illustration 7: Interface avec un sceau chargé

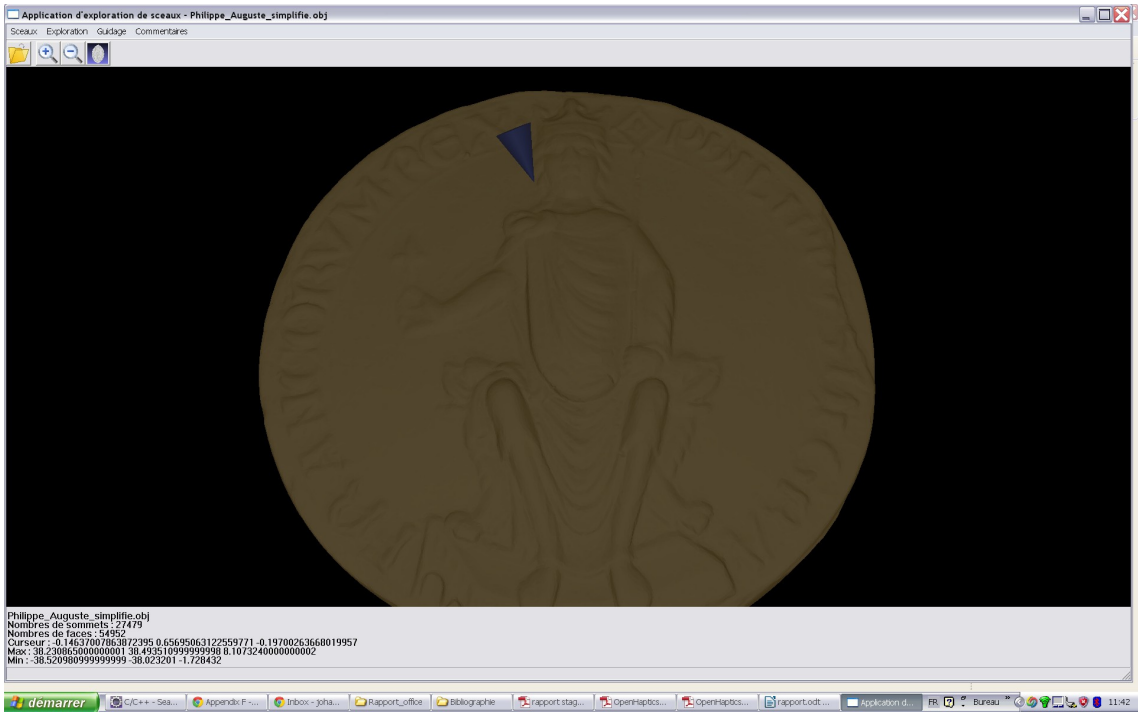

Illustration 8: Interface lors de l'exploration d'un sceau avec le bras haptique (curseur bleu)

3.2.2 OpenGL

Le canevas OpenGL désigne la partie de l'interface graphique dans laquelle on exécute des commandes OpenGL pour afficher des objets 2D ou 3D. Il permet de définir la scène, la position des objets dans la scène et la position de la caméra, c'est-à-dire de quelle manière la scène est visualisée. C'est ici qu'on affiche le sceau et un curseur qui permet de représenter la position du bras haptique dans la scène.

Le système de coordonnées utilisé dans OpenGL est un système main-droite avec l'axe des x pointant à droite, l'axe des y pointant en haut et l'axe des z qui sort de l'écran. L'origine est située en bas à gauche. Les unités sont arbitraires.

OpenGL utilise quatre matrices pour convertir les coordonnées réelles d'un objet en un ensemble de pixels qui s'afficheront sur un écran 2D :

- la **matrice de visualisation** qui permet de positionner la caméra ;

- la **matrice de modélisation** qui permet de transformer les objets avec des rotations, des translations et des mises à l'échelle ;

- la **matrice de projection** qui permet de spécifier de quelle manière est projetée la scène (projection orthographique ou perspective, champ de vision, etc.) ;

- la **matrice *viewport*** qui permet de définir l'espace disponible à l'écran.

Les matrices de visualisation et de modélisation sont regroupées dans une seule matrice pour déplacer et transformer les objets dans la scène.

Pour dessiner un sceau dans le canevas OpenGL, on procède en plusieurs étapes :

1. On définit la région de la fenêtre allouée au canevas en utilisant tout l'espace disponible. Cette opération doit être refaite à chaque redimensionnement de la fenêtre.
2. On définit une projection perspective en précisant le champ de vision et la profondeur de champ.
3. On définit l'emplacement de la caméra et l'angle de vue.
4. On définit et on place une ou plusieurs lumières. Sans lumière les objets n'apparaissent pas. Pour obtenir un rendu plus réaliste, il faut fournir à OpenGL les vecteurs normaux des sommets.
5. On dessine le sceau après lui avoir appliqué diverses transformations. Sa taille est notamment ajustée à celle du volume de visualisation défini à l'étape 2. On peut parcourir toutes les faces du maillage et les dessiner les unes après les autres ou placer les coordonnées des sommets et des

normales aux sommets dans deux tableaux qui sont récupérés par OpenGL pour dessiner les faces. La première méthode requiert deux appels de fonctions par sommet (un pour le sommet, un pour la normale). La seconde ne nécessite que quelques appels de fonctions. Elle est donc beaucoup plus rapide. Mais sur mon ordinateur elle produit des erreurs de segmentation mémoire avec les objets trop gros (plusieurs milliers de sommets).

Les commandes OpenGL qui sont exécutées régulièrement, c'est-à-dire celles qui permettent d'afficher le curseur et le sceau, sont placées dans des *display lists*. Les *display lists* permettent de stocker un groupe de commandes OpenGL dans une liste pour une exécution ultérieure. Cela permet de faire certains calculs sur les matrices et la lumière une seule fois plutôt qu'à chaque exécution. Cependant, à chaque modification du sceau, il faut mettre à jour sa *display list*.

3.3 Interface haptique

Le kit de développement OpenHaptics fournit avec le bras comprend pour chacun des systèmes d'exploitation supportés :

- les pilotes du périphérique ;

- la bibliothèque C **HDAPI (Haptic Device API)** qui fournit un accès bas niveau au bras haptique;

- la bibliothèque C **HLAPI (Haptic Library API)** qui permet un rendu haptique de plus haut-niveau et qui est conçu pour être similaire à OpenGL;

- la bibliothèque C++ **QuickHaptics** qui permet de développer rapidement des interfaces graphique haptiques

- des exemples ;

- de la documentation.

Pour développer l'interface haptique j'utilise essentiellement des fonctionnalités de HLAPI. Les raisons pour lesquelles je n'ai pas retenu QuickHaptics sont expliquées dans le paragraphe 3.4.2. La bibliothèque HDAPI est utilisée pour initialiser le bras et récupérer les dimensions de son espace de travail. La bibliothèque HLAPI permet de définir les paramètres du rendu haptique et d'effectuer ce rendu. Elle est étroitement liée à OpenGL pour localiser les objets et appliquer correctement les collisions entre le bras et les objets.

Il existe deux modes de rendu haptique. Le premier utilise le ***buffer*** **de**

profondeur d'OpenGL. Ce *buffer* stocke la profondeur de chaque pixel affiché à l'écran, c'est-à-dire la distance sur l'axe z de chaque pixel effectivement visible par la caméra OpenGL. Tout ce qui n'est pas visible par la caméra (les parties d'objets cachées par d'autres objets, les faces arrières des objets, etc.) et qui n'apparait pas à l'écran ne peut être récupéré avec ce *buffer*. Par conséquent, il n'est pas possible de ressentir avec le bras ce qui n'est pas visible avec ce type de rendu. Le second mode de rendu utilise directement les primitives OpenGL (sommets, lignes et polygones) via le ***buffer* de *feedback*** d'OpenGL. Avec cette méthode, il est possible de ressentir toutes les faces d'un objet même celles qui ne sont pas visibles et de contraindre le bras à un sommet, une ligne ou une face. Mais les objets sont limités à 65536 sommets. Les sceaux qui ont été retenus contiennent entre 314701 et 675647 sommets. Pour rendre haptiquement ces objets plusieurs solutions sont envisageables :

- déplacer la caméra pour qu'elle visualise les faces cachées. Cette méthode ne permet pas d'avoir accès à l'ensemble de l'objet en haptique à un instant t ;

- simplifier l'objet pour qu'il ne contienne pas plus de 65536 sommets ;

- décomposer l'objet en plusieurs objets qui contiennent 65536 sommets au maximum ;

- et dans le cas particulier d'objets relativement plats comme des sceaux et dont la face arrière ne nous intéresse pas (on peut séparer les sceaux à deux faces en deux sceaux), on peut placer le sceau dans une boite qui permet de simuler un fond et d'encadrer l'exploration en évitant que le bras se perde dans l'espace haptique.

J'utilise la dernière méthode : le rendu haptique du sceau utilise le *buffer* de profondeur alors que celui de la boite utilise directement les faces définies par OpenGL. Mais il est possible d'utiliser une version simplifiée d'un sceau pour le rendu haptique et de conserver la version originale pour le rendu graphique ou d'utiliser dans les deux cas la version simplifiée. En effet, les sceaux ont été acquis avec une résolution très élevée et même avec une simplification qui conserve moins de 10% des sommets, ils conservent leur aspect visuel. cf. codes en annexe pour l'affichage d'un objet.

HLAPI utilise un système de matrices similaires à OpenGL. Pour ajuster l'espace de travail haptique avec l'espace graphique, on réutilise la matrice de projection d'OpenGL. Il est important de noter que l'espace de travail du bras ne correspond pas à tout l'espace disponible, car le bras n'est pas capable de simuler correctement les forces de contact au-delà d'un parallélépipède de 160 mm de large, 120 mm de hauteur et 70 mm de profondeur.

HLAPI utilise également une matrice pour stocker la position et l'orientation du bras. Lors du rendu graphique, cette matrice est multipliée à la matrice de modélisation et visualisation pour afficher un curseur qui se déplace (en

position et en rotation) en suivant les mouvements du bras.

3.4 Importation des sceaux

3.4.1 Formats de fichier

Les sceaux dont je disposais pour effectuer les tests étaient disponibles sous différents formats de fichier de description de géométrie 3D : sous forme textuelle au format OBJ et sous forme binaire aux formats PLY et STL. Comme il est plus facile de manipuler et traiter des données textuelles, j'ai choisi d'utiliser les sceaux sous le format OBJ. En outre, c'est un format de fichier ouvert et largement répandu.

Le format de fichier OBJ est un format ASCII qui utilise une structure similaire à la structure de données *Face-Vertex* présentée au paragraphe .

Elle présente d'abord une liste de sommets avec leurs coordonnées (x,y,z) dans l'espace, puis une liste de faces avec les indices, dans la précédente liste, des sommets qui composent ces faces ordonnés dans le sens trigonométrique. Le format OBJ permet également de préciser une liste des coordonnées de texture et une liste des normales aux sommets. Les définitions des faces précisent alors, en plus des indices des sommets, ceux des textures et des normales. Ces deux dernières informations ne sont pas nécessaires pour représenter un objet en 3D et elles n'apparaissent pas dans les fichiers des sceaux. Cependant, les normales aux sommets sont utilisées par OpenGL lors du rendu graphique des sceaux pour calculer de quelle manière la lumière est renvoyée par chaque sommet. Par conséquent, je les calcule lors de l'importation d'un sceau après le chargement du fichier OBJ. Un exemple de fichier OBJ représentant un cube est disponible en annexe.

Lors de mon travail, j'ai également travaillé avec le format de fichier OFF (Object File Format) car c'est le seul format de description de géométrie 3D supporté par CGAL à la fois en importation et en exportation. Il s'agit d'un format ASCII, ouvert et largement supporté, assez similaire au format OBJ avec une liste des sommets avec leurs coordonnées et une liste des faces avec les indices des sommets. Il est possible de convertir facilement et rapidement des fichiers au format OBJ en format OFF avec le logiciel open-source, gratuit et multiplate-forme Meshlab.

Les fichiers contiennent uniquement les sommets et faces de la surface de l'objet, c'est-à-dire son enveloppe externe. L'intérieur n'est pas défini. Les objets sont donc creux.

3.4.2 Bibliothèque de manipulation de maillage

Dans un premier temps, pour manipuler les sceaux, j'ai utilisé la bibliothèque **QuickHaptics** du SDK OpenHaptics, car elle permet de développer rapidement et simplement des applications haptiques. Elle permet notamment de lire des objets aux formats OBJ, PLY et STL et de les charger sous la forme d'un maillage triangulaire. Cependant, son interface de programmation m'est apparue rapidement trop limitée. Il n'est en effet pas possible de modifier le maillage de l'objet. L'interface ne fournit aucune opération sur les sommets ou les faces et ne définit pas les arêtes. En outre, elle ne permet pas de modifier les coordonnées d'un sommet. Étant donné que le code source de cette bibliothèque est fermé, je ne peux pas rajouter les fonctionnalités dont j'ai besoin.

J'ai donc décidé d'abandonner Quickhaptics et de développer ma propre bibliothèque pour charger des fichiers OBJ dans un maillage triangulaire. J'ai d'abord implémenté une structure de données de type *Face-Vertex*, car les fichiers OBJ présentent une structure similaire. Mais je me suis rapidement aperçue que certaines opérations étaient beaucoup trop lentes avec cette structure. Par exemple, la suppression d'un sommet implique de mettre à jour la liste des faces pour supprimer ou modifier les faces qui contenaient ce sommet. Mais avec une structure de données de type Face-Vertex il n'est pas possible d'accéder facilement aux faces voisines d'un sommet. Dans le pire des cas, il faut parcourir toute la liste des faces pour trouver toutes les faces adjacentes. Cette solution n'est pas efficace avec des objets qui contiennent plusieurs centaines de milliers de faces.

Après quelques recherches sur les structures de données les plus adaptées à mon problème, j'ai choisi d'utiliser ***Half-Edge*** qui est très répandue et bien documentée et qui utilise moins d'espace mémoire que *Winged-Edge*. Dans la structure de données *Half-Edge*, chaque demi-arête stocke un pointeur vers sa demi-arête opposée, son sommet destination, sa face adjacente et la demi-arête suivante dans la face. Chaque sommet stocke ses coordonnées et un pointeur vers l'une des demi-arêtes qui part de ce sommet. Chaque face stocke un pointeur vers une de ses demi-arêtes adjacentes, mais aucune référence vers un quelconque sommet. Pour obtenir la liste des sommets d'une face, on part de sa demi-arête pour obtenir le premier sommet, puis on passe à la demi-arête suivante et ainsi de suite, jusqu'à retomber sur la demi-arête de départ. Pour obtenir la liste des faces adjacentes à un sommet, on part de sa demi-arête pour obtenir la première face, puis on passe à la demi-arête qui suit l'arête opposée et ainsi de suite, jusqu'à retomber sur la demi-arête de départ.

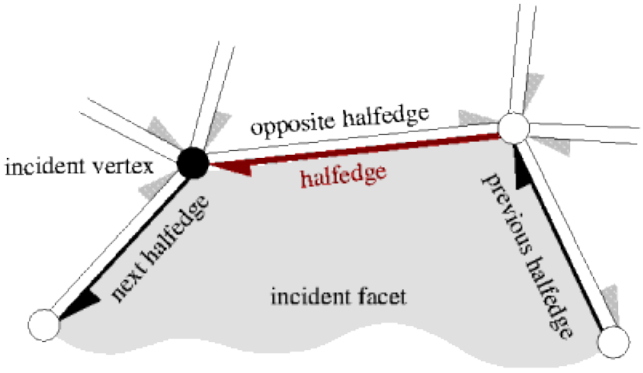

Illustration 9: Structure de données Half-edge

Dans un premier temps, j'ai essayé d'implémenter moi-même cette structure de données. Mais après m'être confrontée à de nombreux problèmes dans la mise à jour des pointeurs, je suis partie à la recherche de bibliothèques capables de gérer ce type de représentation. Les deux bibliothèques les plus mentionnées sont **OpenMesh** [OM] et **CGAL** [CGAL]. Elles sont toutes les deux développées en C++, multi-plateformes et open source. Elles font un usage intensif de la programmation générique, ce qui permet une grande flexibilité dans le type des données utilisées.

La bibliothèque OpenMesh est facile à installer, elle dispose d'une interface de programmation relativement simple et elle est capable de lire et d'écrire des fichiers OBJ. Mais elle ne fournit que la structure de données *Half-Edge*, les méthodes pour parcourir un maillage et ajouter ou supprimer des éléments dans la géométrie du maillage, ainsi qu'un outil de subdivision de maillage.

La bibliothèque CGAL (Computational Geometry Algorithms Library) fournit des structures de données et des algorithmes pour faire des calculs géométriques. Parmi les nombreux outils qu'elle propose, elle implémente la structure de données *Half-edge* pour les maillages triangulaires, les opérations sur les polyèdres (dont la somme de Minkowski), la triangulation de surface, la génération de maillage, la simplification de maillage et l'approximation de crêtes et dans un maillage triangulaire. Cependant, elle ne permet pas de lire ou d'écrire des fichiers OBJ. De plus, son interface de programmation est moins intuitive que celle d'OpenMesh. Malgré tout, elle m'est apparue être la bibliothèque qui offre le plus de possibilités réelles et la moins limitée en terme d’exploitation dans le cadre d’une application.

Pour manipuler la géométrie des sceaux, j'utilise le type *Polyhedron_3* de CGAL

qui permet de représenter un polyèdre sous la forme d'un maillage triangulaire en utilisant la structure de données *Half-Edge*. Ce type est paramétrable. On peut notamment préciser le **noyau** et les **items** utilisés.

Le **noyau** permet de définir :

- le type des données manipulées : nombre à virgule flottante ou nombres rationnels. Dans ce dernier cas, il faut installer une bibliothèque de manipulation de nombres exacts comme GMP (GNU Multiple Precision) ;

- le type de coordonnées utilisées : cartésiennes ou homogènes ;

- l'exactitude des prédicats géométriques utilisés : calculs géométriques exacts, inexacts (moins précis mais beaucoup plus rapides) ou utilisation de filtres arithmétiques qui permettent d'obtenir des résultats exacts plus rapidement qu'avec de simples prédicats géométriques exacts. L'utilisation de prédicats géométriques exacts avec des nombres exacts permet de garantir la robustesse des algorithmes en contrepartie d'un temps de calcul très long.

J'utilise un noyau de type cartésien avec des nombres à virgule flottante à double précision.

Les **items** définissent les sommets, arêtes et faces. J'ai défini mes propres items pour rajouter les normales des faces et des sommets, car les items par défaut ne les contiennent pas.

J'utilise le type ***Polyhedron_3*** dans la classe Mesh de la bibliothèque de manipulation de maillage que j'ai développée. Cette classe permet de lire des fichiers OFF et des fichiers OBJ. Si CGAL ne permet pas de lire directement des fichiers au format OBJ, elle fournit une classe paramétrable pour construire un *Polyhedron_3* de façon incrémentale. J'utilise cette classe pour lire un fichier OBJ et remplir une liste de sommets et une liste de faces qui sont ensuite utilisées pour construire le polyèdre. La classe Mesh fournit des méthodes pour modifier le maillage et pour récupérer les dimensions de la boite englobante et de son centre.

Dans l'application d'exploration, la classe Sceau permet de faire le lien entre les commandes du menu de l'interface et les méthodes implémentées dans la classe Mesh.

3.5 Zoom

L'application permet d'afficher les sceaux à plusieurs échelles pour avoir une vue globale du sceau ou percevoir certains détails avec plus de précision. Les différentes fonctions de zoom qui ont été implémentées sont :

- le **zoom à taille réelle** qui permet d'afficher le sceau dans ses dimensions

réelles ;

- le **zoom écrin** qui permet d'ajuster la taille du sceau aux dimensions du volume de visualisation et qui pivote le sceau sur l'axe des x pour permettre une meilleure exploration (par défaut les sceaux sont affichés verticalement) ;

- des fonctions de **zoom** qui permettent d'augmenter et de diminuer l'échelle du sceau dans la limite des dimensions du volume de visualisation.

Pour réaliser ces mises à l'échelle, on peut modifier la matrice de modélisation des sceaux ou modifier directement le sceau. J'ai développé une fonction dans la classe Mesh qui permet de changer l'échelle d'un maillage. Mais elle nécessite de parcourir tous les sommets du maillage, ce qui peut être assez long sur les gros modèles. J'utilise donc des fonctions de transformation d'OpenGL qui modifie la matrice de modélisation des sceaux. Il est possible d'utiliser des échelles différentes pour chacun des axes.

3.6 Simplification

La bibliothèque CGAL propose une implémentation d'un algorithme de simplification de maillages triangulaires. Cet algorithme utilise un type particulier d'effondrement d'arêtes : l'effondrement de demi-arêtes. Alors que l'effondrement d'une arête supprime les deux sommets de l'arête et crée un nouveau sommet, l'effondrement d'une demi-arête fusionne l'un des sommets de l'arête dans l'autre sommet. Dans l'algorithme proposé, il est également possible de déplacer le sommet restant (ce qui revient à faire de l'effondrement d'arête).

Les différents paramètres de cet algorithme sont :

- le **maillage à simplifier**. Il doit être obligatoirement fourni par l'utilisateur et doit supporter l'effondrement d'arêtes. La structure de données que j'utilise, l'*Half-edge*, supporte l'effondrement d'arêtes.

- la **condition d'arrêt**. Elle doit être obligatoirement fournie par l'utilisateur. Il s'agit soit du nombre d'arêtes soit du pourcentage d'arêtes que l'on souhaite préservées dans le maillage simplifié.

- la **fonction de coût**. Elle permet de calculer le coût d'un effondrement. CGAL propose deux fonctions de coût. La première utilise les travaux de P. Lindstrom et G. Turk [19]. Dans la seconde, le coût de l'effondrement d'une arête est le carré de sa longueur. Elle est plus rapide mais moins précise.

- la **fonction de placement**. Elle détermine la position du nouveau sommet. CGAL propose deux fonctions de placement. La première utilise les travaux de P. Lindstrom et G. Turk [19]. La deuxième place simplement le nouveau

sommet au milieu de l'arête effondrée. Elle est plus rapide, mais moins précise.

- un visiteur. Ce paramètre est optionnel. C'est un objet qui permet de suivre le processus de simplification (par exemple, afficher les arêtes qui sont effondrées).

Par défaut, si aucun paramètre n'est précisé pour les fonctions de coût et de placement, CGAL utilise celles de P. Lindstrom et G. Turk. Il est possible d'implémenter ses propres fonctions de coût et de placement et de les passer en paramètres à l'algorithme de CGAL. On peut également marquer les arêtes qui ne doivent pas être supprimées avant la simplification.

Pour l'instant, j'utilise simplement l'algorithme avec les paramètres par défaut. Sur un modèle comprenant environ 3000 sommets, l'algorithme met environ six secondes pour appliquer une simplification qui conserve la moitié des sommets. La même opération sur un sceau d'environ 315000 sommets prend environ 11 minutes.

Illustration 10: Modèle 3D original d'une vache (2904 sommets)

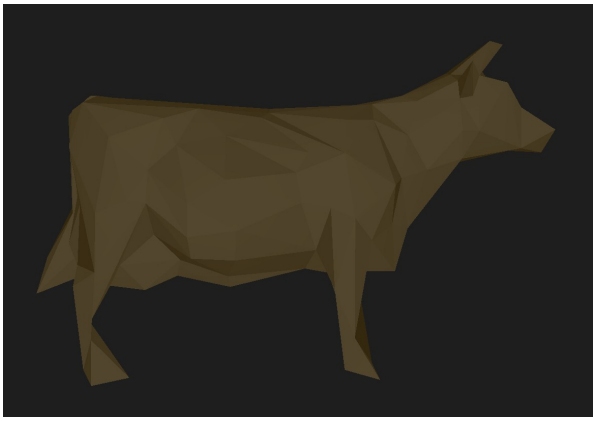

Illustration 11: Le même modèle après quatre applications successives de l'algorithme de simplification (182 sommets)

3.7 Filtres

J'ai commencé à implémenter des filtres simples qui permettent de supprimer un sommet si sa coordonnée sur l'axe z est en-dessous ou au-dessus d'un certain seuil. Dans CGAL, pour changer la géométrie d'un *Polyhedron_3*, il faut dériver une classe abstraite paramétrable qui fournit l'interface pour n'importe quelle modification. Il faut également mettre à jour soi-même les sommets, les faces et les arêtes qui ont été modifiées par le changement que l'on vient d'appliquer. Actuellement, j'ai encore des problèmes au niveau de la mise à

jour correcte du maillage après la suppression du sommet.

3.8 Morphologie mathématique

La bibliothèque CGAL fournit la somme de Minkowski de deux polygones ou de deux polyèdres. Cet algorithme ne peut être appliqué que sur des polygones et des polyèdres Nef [NEF]. CGAL dispose d'une interface pour convertir un polyèdre en polyèdre Nef à condition que les noyaux utilisés par les deux types de polyèdres soient compatibles. En particulier, les calculs sur les polyèdres Nef nécessitent la manipulation de nombres exacts et donc un noyau qui utilise des nombres exacts. Il est également possible de charger un fichier OFF dans un polyèdre Nef.

Sous Linux, le calcul de la somme de Minkowski d'une sphère et d'un cube prend plusieurs minutes. Sous Windows, l'application ne compile plus, car les fichiers générés par CGAL sont trop gros pour le compilateur que j'utilise.

4 Conclusion

Au cours de ce stage, j'ai mis au point une première version de l'application d'exploration des sceaux qui fonctionne sous Windows et sous Linux et qui servira de base à des développements et des expérimentations futures. J'ai développé l'interface, le chargement, le rendu graphique et haptique des sceaux et j'ai commencé à développer certaines transformations. Je n'ai pas pu avancer autant que je le souhaitais sur les transformations mais j'ai pu mettre en place le cadre dans lequel les algorithmes s'intégreront. Il reste maintenant à continuer le développement des transformations, à intégrer le développement concernant le guidage et les commentaires audio et à tester l'application.

Pendant mes recherches et la programmation de l'application, j'ai été confrontée à plusieurs difficultés. Quand j'ai entamé ce stage, je pensais que les outils de développement fournis avec le bras disposaient d'une interface de programmation pour accéder à la géométrie des modèles 3D. En fait, ces outils sont trop limités pour répondre seuls aux objectifs du projet. En outre, il n'existe pas de bibliothèques logicielles de haut niveau développées pour le bras qui répond à nos besoins. J'ai ensuite tâtonné plusieurs semaines avant de trouver une bibliothèque qui me permettait d'appliquer des algorithmes géométriques complexes à des modèles 3D. J'ai aussi rencontré des difficultés pour créer la scène avec OpenGL et placer les objets. Il est en effet très facile de produire des dizaines de lignes de codes qui aboutissent à un simple écran noir et si les objets n'apparaissent pas, ils ne peuvent pas être perçus avec le bras puisque l'espace haptique est très fortement lié à l'espace graphique. Enfin, mes modestes et lointaines connaissances en mathématiques et en traitement d'images ont rendu la lecture et la compréhension des articles de recherches et des algorithmes géométriques parfois ardues.

Malgré tout, ce stage m'a permis de découvrir un domaine passionnant dans lequel beaucoup reste à faire. Il m'a permis de confirmer mon intérêt pour le domaine du handicap et la conception de logiciels visant à améliorer l'accès aux informations et aux savoirs pour tous.

5 Bibliographie

[1] GENTAZ, Edouard et HATWELL, Yvette. Haptic processing of spatial and material object properties. *Touching for knowing*, 2003, p. 123-159.

[2] GENTAZ, Edouard. Caractéristiques générales de l'organisation anatomo-fonctionnelle de la perception cutanée et haptique. In : Y. Hatwell, A. Streri & E. Gentaz (Eds.), *Toucher pour connaître. Psychologie cognitive de la perception tactile manuelle*. Paris: Presses Universitaires de France. (2000) : p. 19-34.

[3] LEDERMAN, Susan J. et KLATZKY, Roberta L. Hand movements: A window into haptic object recognition. *Cognitive psychology*, 1987, vol. 19, no 3, p. 342-368.

[4] LEDERMAN, Susan J. et KLATZKY, Roberta L. Extracting object properties through haptic exploration. *Acta psychologica*, 1993, vol. 84, no 1, p. 29-40.

[5] JANSSON, Gunnar, PETRIE, H., COLWELL, C., *et al.* Haptic virtual environments for blind people: exploratory experiments with two devices. *International Journal of Virtual Reality*, 1999, vol. 4, no 1, p. 10-20.

[6] JANSSON, G. Basic issues concerning visually impaired people's use of haptic displays. In : *Proc. 3rd International Conference on Disability, Virtual Reality and Associated Technologies*. 2000. p. 33-38.

[7] JANSSON G. Perceiving complex virtual scenes without visual guidance. In: McLaughlin M.L., Hespanha J., Sukhatme G., editors. *Touch in virtual environments*. Upper Saddle River, NJ: Prentice Hall. 2002. p. 169–179.

[8] JANSSON, Gunnar, BERGAMASCO, Massimo, et FRISOLI, Antonio. A new option for the visually impaired to experience 3D art at museums: Manual exploration of virtual copies. *Visual Impairment Research*, 2003, vol. 5, no 1, p. 1-12.

[9] KLATZKY, Roberta L. et LEDERMAN, Susan J. Tactile roughness perception with a rigid link interposed between skin and surface. *Perception & psychophysics*, 1999, vol. 61, no 4, p. 591-607.

[10] COLWELL, Chetz, PETRIE, Helen, KORNBROT, Diana, *et al.* Haptic virtual reality for blind computer users. In : *Proceedings of the third international ACM conference on Assistive technologies*. ACM, 1998. p. 92-99.

[11] BAUMGART, Bruce G. A polyhedron representation for computer vision. In :*Proceedings of the May 19-22, 1975, national computer conference and exposition*. ACM, 1975. p. 589-596.

[12] KETTNER, Lutz. Using generic programming for designing a data structure for polyhedral surfaces. *Computational Geometry*, 1999, vol. 13, no 1, p. 65-90.

[13] GUIBAS, Leonidas et STOLFI, Jorge. Primitives for the manipulation of general subdivisions and the computation of Voronoi. *ACM Transactions on Graphics (TOG)*, 1985, vol. 4, no 2, p. 74-123.

[14] ROSSIGNAC, Jarek et BORREL, Paul. *Multi-resolution 3D approximations for rendering complex scenes*. Springer Berlin Heidelberg, 1993.

[15] SCHROEDER, William J., ZARGE, Jonathan A., et LORENSEN, William E. Decimation of triangle meshes. In : *ACM SIGGRAPH Computer Graphics*. ACM, 1992. p. 65-70.

[16] HOPPE, Hugues. Progressive meshes. In : *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996. p. 99-108.

[17] GARLAND, Michael et HECKBERT, Paul S. Surface simplification using quadric error metrics. In : *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1997. p. 209-216.

[18] MATIAS VAN KAICK, Oliver et PEDRINI, Hélio. A comparative evaluation of metrics for fast mesh simplification. In : *Computer Graphics Forum*. Blackwell Publishing Ltd, 2006. p. 197-210.

[19] LINDSTROM, Peter et TURK, Greg. Fast and memory efficient polygonal simplification. In : *Visualization'98. Proceedings*. IEEE, 1998. p. 279-286.

[20] LINDSTROM, Peter et TURK, Greg. Evaluation of memoryless simplification.*Visualization and Computer Graphics, IEEE Transactions on*, 1999, vol. 5, no 2, p. 98-115.

[21] BEUCHER S. et LANTUEJOUL C. Use of watersheds in contour detection. In : *International Workshop on image processing, real-time edge and motion detection/estimation*, Rennes, France, Sept. 1979.

[22] MANGAN, Alan P. et WHITAKER, Ross T. Partitioning 3D surface meshes using watershed segmentation. *Visualization and Computer Graphics, IEEE Transactions on*, 1999, vol. 5, no 4, p. 308-321.

[23] ATTENE, Marco, KATZ, Sagi, MORTARA, Michela, *et al.* Mesh segmentation-a comparative study. In : *Shape Modeling and Applications, 2006. SMI 2006. IEEE International Conference on*. IEEE, 2006. p. 7-7.

Sites internet

[OH] OpenHaptics : http://support.sensable.com/support-openhaptics.htm

[WX] wx-widgets : http://www.wxwidgets.org

[OM] OpenMesh :http://www.openmesh.org

[CGAL] CGAL :http://www.cgal.org

[BOOST] Boost : http://www.boost.org

[NEF] Nef Polyhedron (définition en anglais) :http://en.wikipedia.org/wiki/Nef_polygon

6 Annexes

6.1 Exemple de fichier OBJ représentant un cube

```
# cube.obj

g cube

# liste des sommets avec leurs coordonnées
v  0.0  0.0  0.0
v  0.0  0.0  1.0
v  0.0  1.0  0.0
v  0.0  1.0  1.0
v  1.0  0.0  0.0
v  1.0  0.0  1.0
v  1.0  1.0  0.0
v  1.0  1.0  1.0

# liste des normales aux sommets
vn  0.0  0.0  1.0
vn  0.0  0.0 -1.0
vn  0.0  1.0  0.0
vn  0.0 -1.0  0.0
vn  1.0  0.0  0.0
vn -1.0  0.0  0.0

# liste des faces avec
# les indices des sommets
# les indices des normales
f  1//2  7//2  5//2
f  1//2  3//2  7//2
f  1//6  4//6  3//6
f  1//6  2//6  4//6
f  3//3  8//3  7//3
f  3//3  4//3  8//3
f  5//5  7//5  8//5
f  5//5  8//5  6//5
f  1//4  5//4  6//4
f  1//4  6//4  2//4
f  2//1  6//1  8//1
f  2//1  8//1  4//1
```

6.2 Interface de la classe Mesh

```
* mesh.hh
 */

#ifndef MESH_HH_
#define MESH_HH_

#include "my_polyhedron_items_3.hh"
#include "my_polyhedron_builder.hh"

#include <CGAL/Simple_cartesian.h>  // use cartesian coordinates to represent
                                    // geometric objects
#include <CGAL/Polyhedron_3.h>      // use HE structure to represent polyhedral
                                    // surfaces
#include <CGAL/IO/Polyhedron_iostream.h>

#include <string>

typedef CGAL::Simple_cartesian<double>                              Kernel;
typedef CGAL::Polyhedron_3<Kernel, My_polyhedron_items_3>           Polyhedron;
typedef Polyhedron::HalfedgeDS                                      HDS;
typedef Polyhedron::Halfedge_handle                                 HE_handle;
typedef Kernel::Point_3                                             Point_3;

namespace MeshProcessing {

      typedef Kernel::Point_3 MP_vec3d;

      class Mesh {
      public:
            // ctors and dtors
            Mesh();
            Mesh(std::string filename);
            ~Mesh();

            // getters and setters
            Kernel::Iso_cuboid_3 getBoundingBox();
            MP_vec3d getBoundingBoxMax(); // back, top, right
            MP_vec3d getBoundingBoxMin(); // front, bottom, left
            MP_vec3d getCentroid();
            unsigned int getNumVertices();
            unsigned int getNumFaces();

            // transformation methods
            void scale(double fx, double fy, double fz);
            void slice(double zMax, double zMin);
            void lowPassFilter(double zMax);
            void highPassFilter(double zMin);
            void simplify(float r);

            void dilatation(std::string filename);

      private:
            Polyhedron P;
            Kernel::Iso_cuboid_3    bbBox;
            Kernel::Point_3         centroid;
      };
```

```
}

#endif /* MESH_HH_ */
```

6.3 Interface de la classe Sceau

```
/* seal.hh
 */

#ifndef SEAL_HH_
#define SEAL_HH_

#include <mesh.hh>
#include <GL/gl.h>

#include <string>

#define NB_MAX_RENDER_LISTS         10
#define NB_MAX_SIMPLIFICATION       4

class Sceau {
public:
      // ctors and dtors
      Sceau();
      Sceau(std::string name, std::string pathname, double aspect);
      ~Sceau();

      // getters and setters
      std::string getName();
      std::string getPathname();
      double* getTranslation();
      void setPathname(std::string newPathname);
      double getXMax();
      double getYMax();
      double getZMax();
      double getXMin();
      double getYMin();
      double getZMin();
      double* getCentroid();

      // render methods
      void renderGraphics();
      void renderHaptics(unsigned int shapeId, unsigned int ringId);

      // wrapper methods
      void simplify();
      void lowPassFilter();
      void highPassFilter();
      void dilatation();
      void zoom(double z);
      void zoom(double z[3]);
      void realSize();
      void zoomInt(double aspect);
      void zoomEcrin();

      MeshProcessing::Mesh mesh;
```

```
private:
      std::string       name;
      std::string       pathname;
      double            scale[3];
      double            rotationAngle[3];
      int               rotationVector[3];
      double            translation[3];
      int               nbSimplification;
      const int         nbMaxRenderLists;
      GLuint            currentList;
      GLuint            renderGList[NB_MAX_RENDER_LISTS]; // display lists index
      /* TODO
       * ajouter la liste des éléments (et/ou parties)
       * ajouter la liste des commentaires
       */
};

#endif /* SEAL_HH_ */
```

6.4 Interface de la classe Device

```
/* device.hh
 */

#ifndef DEVICE_HH_
#define DEVICE_HH_

#include <HD/hd.h>
#include <HL/hl.h>
#include <HLU/hlu.h>

class Phantom {
public:
      // ctors and dtors
      Phantom();
      ~Phantom();

      // misc
      bool init();

      static const int  cursorPixelSize = 100;
      double            cursorScale;
      HHD               handle;
      HHLRC             renderContext;
      int               cursorList;
      HLdouble          proxyPos[3];
      HDdouble          usableWorkspace[6];
      HDdouble          maxWorkspace[6];
      HLdouble          workspaceDims[6];
      HLuint            myShapeId;
      HLuint            ringId;
};


#endif /* DEVICE_HH_ */
```