Rapport de stage M2 PRO - IICAO[1]

Année universitaire 2007 - 2008 (stage du 1 avril au 31 septembre 2008)

“Modélisation physique interactive d’un genou humain”

sous la tutelle de **François Faure** (laboratoire Évasion) et
François Boux de Casson (société Aesculap), au sein de l’équipe **Évasion**

Vincent Vansuyt

[1] Master professionnel “Mathématiques, Informatique” spécialité : Ingénierie de l’image et de la CAO

Remerciements

Je remercie tous les membres de l'équipe ÉVASION de m'avoir accueilli pendant ce stage.

Je remercie particulièrement François Faure et François Boux de Casson de m'avoir permis de travailler sur ce sujet. Merci également de leur aide pour sa réalisation.

Je remercie spécialement (et chronologiquement) ceux qui ont contribué à l'avancement de ce projet :
- Merci à Anne Pierson de son efficacité lors du traitement de mon dossier un peu particulier.
- Merci à Florent Falipou et Michaël Adam de la résolution de tous mes problèmes liés à SOFA et de leur patience.
- Merci à Noura Faraj et Guillaume Bousquet de leur aide au début du stage et de leur bonne humeur.
- Merci au docteur Olivier Palombi d'avoir développé la base myCorporisFabrica, de ses remarques médicales, de l'obtention d'un IRM haute résolution à l'hôpital sud et de sa gentillesse.
- Merci au docteur Jean-Noël Ravey et à son équipe de nous avoir fourni des données IRM de qualité lors d'un examen à l'hôpital sud, au mois de juin.
- Merci à Antonin Fontanille d'avoir permis de numériser son genou lors de l'IRM.
- Merci à Xiaomao Wu de la démonstration du logiciel Amira.
- Et merci à A.I. Kapandji [1] qui m'a aidé indirectement avec son ouvrage de référence sur le genou, presque indispensable pour ce stage.

Table des matières

Chapitre 1

Introduction

1.1 Sujet du stage et contexte de travail

Ce stage s'effectue au sein de l'équipe Évasion (www-evasion.imag.fr) à l'INRIA Rhône-Alpes et s'appuie sur la librairie logicielle SOFA (www.sofa-framework.org) développée en C++/OpenGL. Il est co-encadré par François Faure, de l'équipe Évasion, et François Boux de Casson, de la société Aesculap.

1.1.1 Objectif

L'objectif de ce travail est de proposer une modélisation informatique interactive d'un genou humain. Le genou est une articulation cruciale physiologiquement, sa structure complexe est adaptée aux nombreuses contraintes auxquelles cette articulation est soumise. Elle gouverne la cinématique du tibia et de la fibula par rapport au fémur et doit être stable en station verticale statique, durant la marche, la course et les sauts. Les traumatismes et l'usure du genou se traitent dans certains cas sévères par la chirurgie orthopédique. Le modèle interactif pourrait être utilisé dans un premier temps pour calculer la cinématique de l'articulation en fonction des sites d'implantation du greffon, dans un scénario de greffe du ligament croisé antérieur. S'il est validé, le modèle pourrait être ensuite utilisé pour simuler la cinématique de genoux équipés de prothèses uni ou bi-compartimentales.

1.1.2 Le modèle à construire

Le modèle sera composé de corps rigides et de corps déformables, reliés ou non entre eux par des contraintes mécaniques de différentes natures.

1.1.3 Présentation de la société Aesculap

Aesculap est une division du groupe B.Braun. Ce groupe est spécialisé dans la conception, la production, la commercialisation et la distribution de matériel médico-chirurgical et de produits pharmaceutiques.
En 2007, les entreprises du groupe ont réalisé un chiffre d'affaires de plus de 3 milliards d'euros avec un effectif de plus de 35 000 collaborateurs, autour de l'activité des produits de santé.
Le groupe B. Braun est l'un des leaders mondiaux dans son secteur d'activité.

FIG. 1.1 – Pays où BBraun est implanté

Aesculap développe des produits et services qui couvrent l'intégralité des besoins du bloc opératoire (instruments de chirurgie, moteurs chirurgicaux, containers de stérilisation, optiques, lumière froide, caméras, vidéo numérique, insufflateurs, instrumentation pour chirurgie endoscopique, prothèses de parois, pinces à clips, à chargeurs, prothèses vasculaires, patchs vasculaires, filtres à veine cave, prothèses de hanche, de genou, et de rachis, navigation, traumatologie, substituts osseux, instruments vétérinaires, tondeuses, implants et sutures spécialisés pour petits et gros animaux, usage unique et solutés, instrumentation pour chirurgie dentaire, ...).
Le centre "R&D Navigation" Aesculap Grenoble se situe près de l'hôpital sud et s'occupe principalement du développement du logiciel Orthopilot.

FIG. 1.2 – Utilisation de produits Aesculap en bloc opératoire

L'Orthopilot est un système de navigation pour l'orthopédie. Ce système permet de positionner précisément les prothèses de genou, de hanche ou de pratiquer des reconstructions de ligaments croisés antérieurs.

FIG. 1.3 – Vue du logiciel Orthopilot (au fond) et de ses périphériques de localisation (dans les mains des chirurgiens)

1.1.4 Présentation du laboratoire Évasion

L'INRIA est l'Institut National pour la Recherche en Informatique et en Automatique. L'inria a fêté ses 40 ans en 2007.

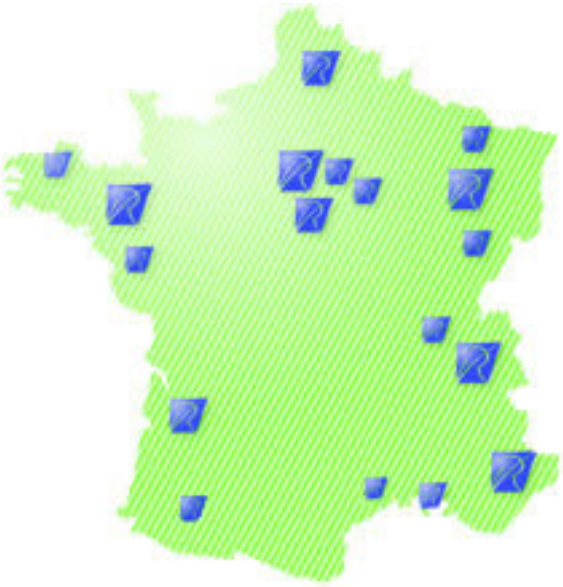


FIG. 1.4 – Implantation des centres Inria en France

Les locaux de l'Inria sont situés à Montbonnot, entre la Charteuse et l'autoroute.

FIG. 1.5 – Les locaux de l'Inria à Montbonnot

L'Inria Grenoble regroupe 27 équipes de recherche qui travaillent sur les thèmes suivants :
- Thème Bio : Systèmes biologiques
- Thème Cog : Systèmes cognitifs
- Thème Com : Systèmes communicants
- Thème Num : Systèmes numériques
- Thème Sym : Systèmes symboliques

L'équipe Évasion fait partie des systèmes cognitifs avec les équipes suivantes :
- ARTIS - Acquisition, représentation et transformations pour l'image de synthèse
- ÉVASION - Environnements virtuels pour l'animation et la synthèse d'images d'objets naturels
- I3D - Interaction 3 dimensions
- LEAR - Apprentissage et reconnaissance en vision par ordinateur
- MISTIS - Modélisation et Inférence de phenomenes aléatoires complexes et structures
- PERCEPTION - Interprétation et Modélisation d'Images et de Vidéos
- PRIMA - Perception, reconnaissance et intégration pour la modélisation des activités

Jusqu'en 2002 l'équipe Évasion s'appelait "Imagis" (Modèles, Algorithmes, Géométrie pour le Graphique et l'Image de Synthèse), avant d'être remplacée par Artis et Évasion.

Les axes de recherche de cette équipe sont :
- Le développement d'outils fondamentaux
  - Spécification de scènes et objets naturels
  - Modèles alternatifs pour la forme, le mouvement et l'apparence
  - Algorithmes adaptatifs et niveaux de détail
- Étude de scènes naturelles spécifiques et applications
  - Scènes minérales (océans, ruisseaux, laves, avalanches, nuages)
  - Scènes végétales (morphogénèse de plantes, prairies, arbres)
  - Monde animal (simulation d'organes, visages et corps d'un personnage, mouvements d'animaux)

Voici-ci dessous, figures 1.6 à 1.13, quelques unes des magnifiques publications de l'équipe Évasion :

FIG. 1.6 – Rendu de nuages, en temps réel

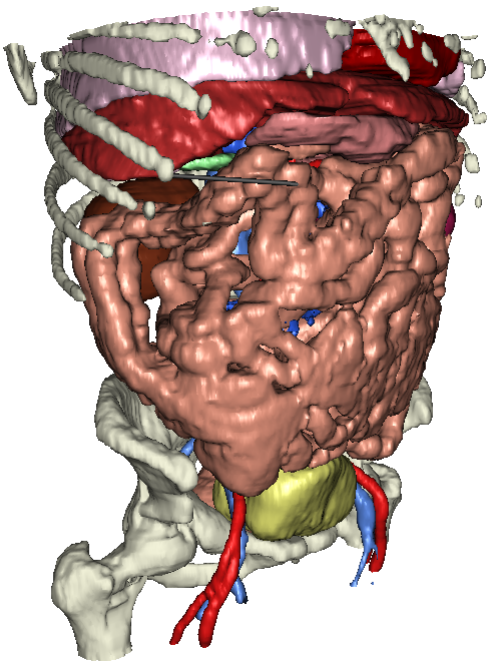


FIG. 1.7 – Scène médicale (avec SOFA)

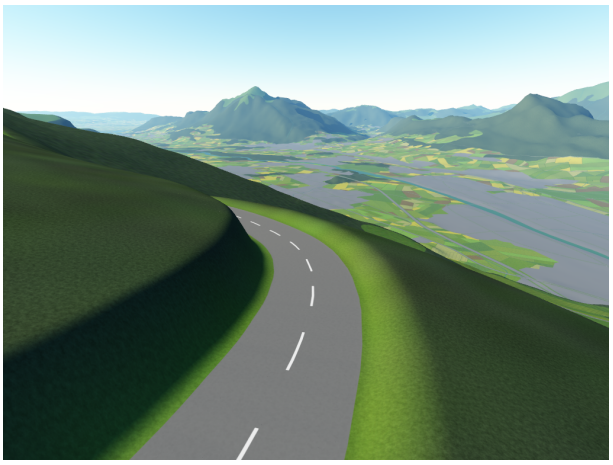


FIG. 1.8 – Rendu de terrains vectoriels, en temps réel

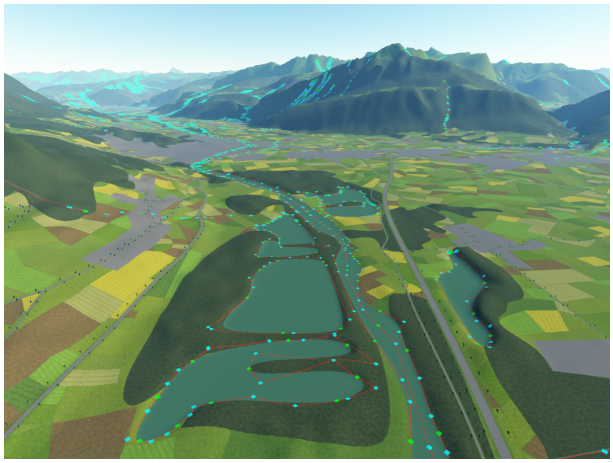


FIG. 1.9 – Rendu de terrain vectoriel, en mode édition

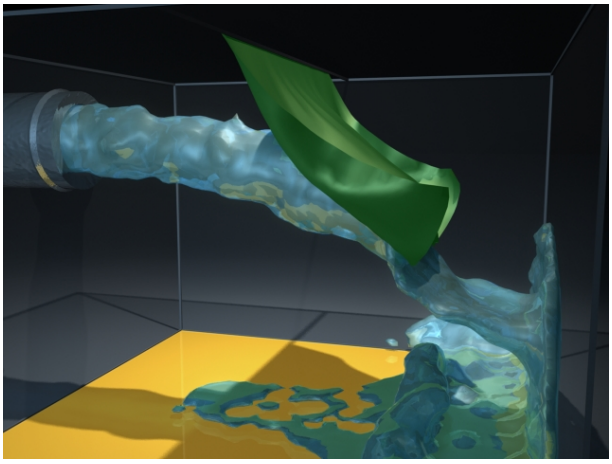


FIG. 1.10 – Fluides et collisions (avec SOFA)

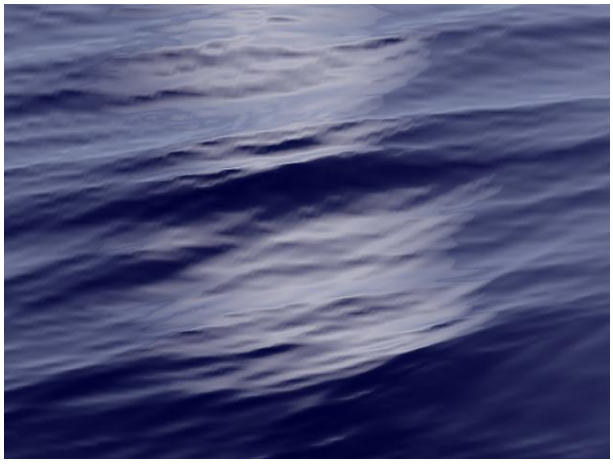


FIG. 1.11 – Vagues de l'océan

FIG. 1.12 – Cheveux

FIG. 1.13 – Objets hautement déformables

1.2 La plateforme SOFA

1.2.1 Présentation

SOFA (Simulation Open Framework Architecture) est une librairie open source de simulation principalement temps réel. Elle est développée par des équipes françaises de l'Inria (équipes ALCOVE[1], ASCLEPIOS[2] et ÉVASION[3]) en collaboration avec des partenaires internationaux :
– CIMIT (USA).
– CSIRO (Australie).
– D'autres équipes en Chine, au Royaume-Uni et en Allemagne.
Ce logiciel peut être utilisé comme librairie ou avec une interface graphique par défaut. SOFA réussit le challenge de pouvoir être utilisé aussi bien dans des projets de jeu vidéo que dans des développements de simulations médicales.

Le but de SOFA est de (d'après la plaquette de présentation distribuée aux journées SOFA) :
– simplifier le développement de simulateurs médicaux en améliorant l'inter-opérabilité des algorithmes,
– promouvoir la collaboration entre groupes de recherche,
– faciliter le transfert de technologie entre recherche et industrie,
– évaluer et valider de nouveaux algorithmes,
– accélérer le prototypage de simulateurs en améliorant la réutilisabilité des composants.

[1] Lille
[2] Sofia Antipolis
[3] Grenoble

La liste des composants utilisables avec SOFA est impressionnante. En voici quelques exemples (voir figures 1.14 à 1.22).

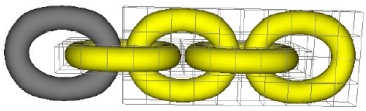


FIG. 1.14 – Composant “Regular grid spring force field”

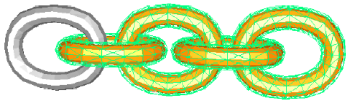


FIG. 1.15 – Composant “Mesh spring force field”

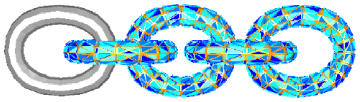


FIG. 1.16 – Composant “Tetrahedron FEM Force Field”

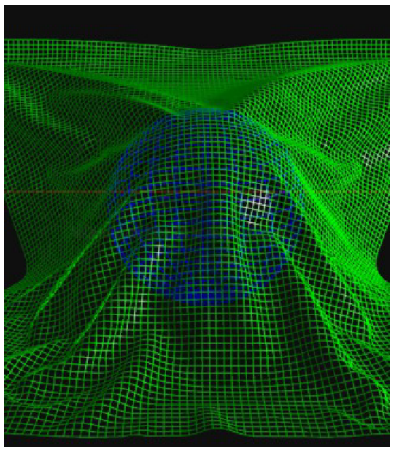


FIG. 1.17 – Composant “Quad Bending Springs”

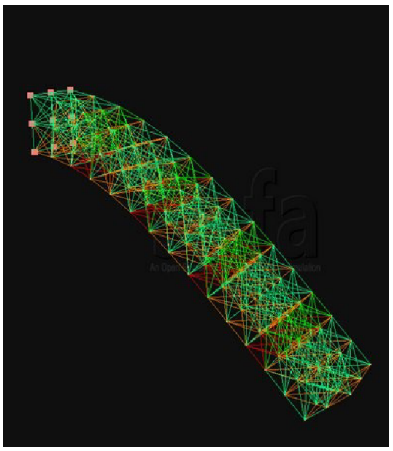


FIG. 1.18 – Composant “Stiff Spring Force Field”

FIG. 1.19 – Gestion des contacts par pénalités

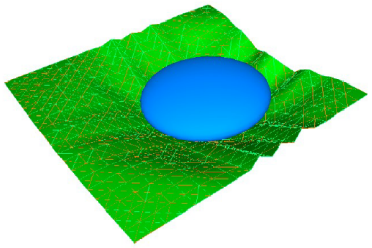


FIG. 1.20 – Champ de force implicite

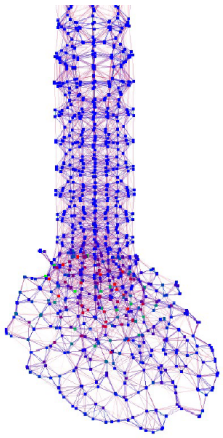


FIG. 1.21 – Composant “Smooth Particule Hydraulics”

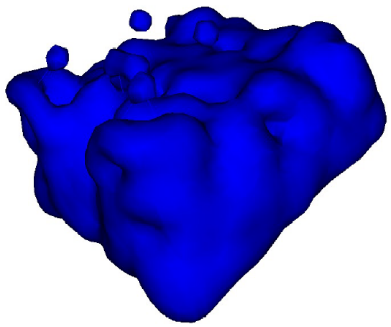


FIG. 1.22 – Composant “Lennard-Jones force field”

Tous ces composants peuvent être utilisés en même temps dans une même scène et interagir. Il est possible, par exemple, de créer des interactions entre des fluides, des objets déformables et/ou des objets rigides tout en utilisant un grand pas de temps et des raideurs importantes.

1.2.2 Exemple de scènes réalisées avec SOFA

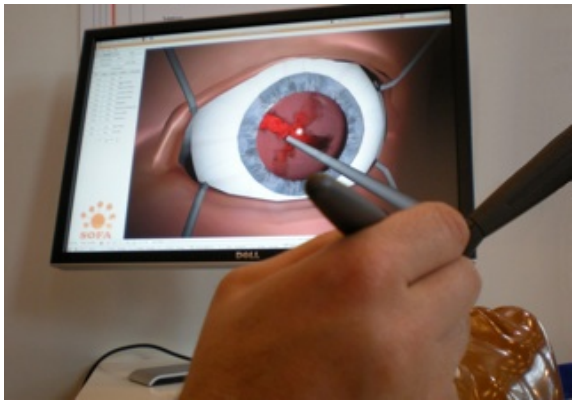


FIG. 1.23 – Simulation médicale avec SOFA

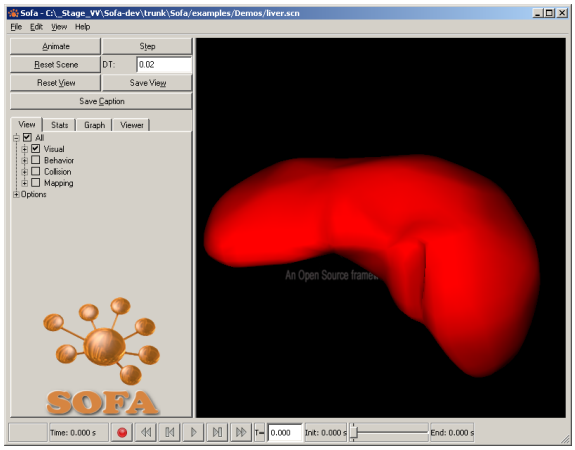


FIG. 1.24 – Exemple d'un foie et interface de SOFA

1.2.3 Mode emploi

Pour créer une scène dans SOFA, il suffit de créer un arbre composé seulement de deux types d'éléments : des noeuds et des objets. Cet arbre peut être créé directement en C++ ou avec un fichier XML au format SOFA, comme sur la figure 1.25.

```
<Node name="root">
    <Object type="..." ...>
    <Object type="..." ...>
    <Object type="..." ...>
    <Node name="...">
        <Object type="..." ...>
        <Object type="..." ...>
        <Node>
            <Object type="..." ...>
            <Object type="..." ...>
        </Node>
        <Node>
            <Object type="..." ...>
        </Node>
    </Node>
    <Node>
        <Object type="..." ...>
        <Object type="..." ...>
    </Node>
</Node>
```

FIG. 1.25 – Exemple d'arbre de scène XML SOFA (incomplet)

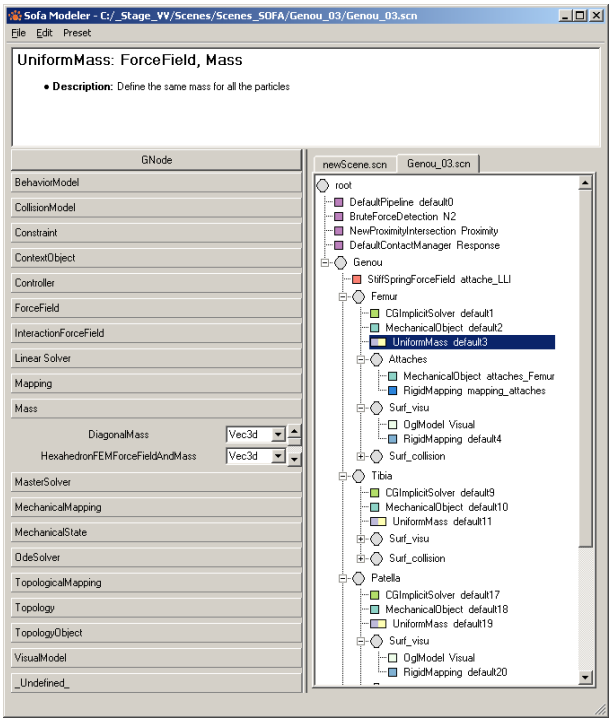


FIG. 1.26 – Copie d'écran du "modeleur"

La construction de l'arbre XML SOFA peut-être assistée par le Modeleur (figure 1.26), un outil créé pendant le stage par Florent Falipou. Ce programme permet de construire un arbre de scène XML-SOFA facilement et de remplir les nombreux attributs des noeuds de type "Object" en fonction de leur nom de "type".

Une fois la scène construite, il suffit d'ouvrir l'exécutable "runSofa.exe" (la figure 1.27 apparaît, avec la scène du foie par défaut), puis dans le menu "File - open", choisir le nom du fichier scène.

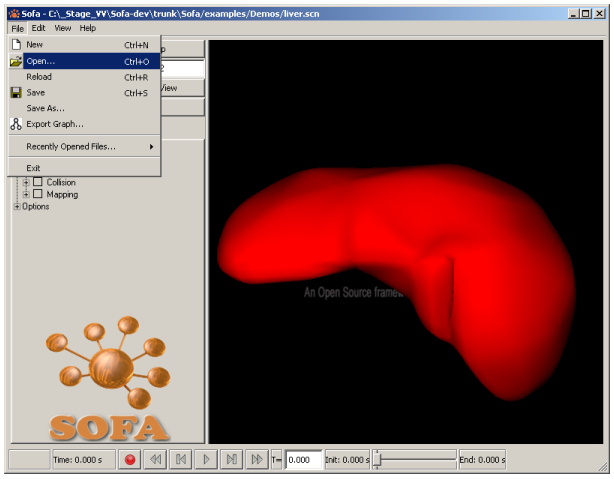


FIG. 1.27 – Ouverture d'une scène SOFA

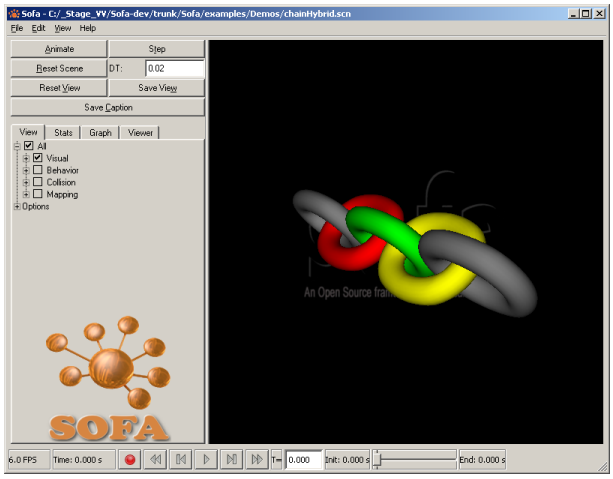


FIG. 1.28 – Ouverture de l'exemple chainHybrid

La scène choisie (il en existe de nombreuses livrées par défaut avec SOFA) apparaît alors dans la fenêtre noire de SOFA (figure 1.28) et l'appui sur le bouton "Animate" lance la simulation.

Cet exécutable ("runSofa") livré avec SOFA et est disponible sur la page http://www.sofa-framework.org/, dans la section "Download" (il est disponible pour tous les OS courants [Mac, Linux, Windows]) et vous pouvez le télécharger et l'essayer facilement.

La version pour Windows est livrée avec un installeur et si vous avez dix minutes, vous pouvez déjà facilement jouer avec une scène SOFA, sans aucune connaissance particulière !

1.3 Anatomie du genou

1.3.1 Plans anatomiques

Voici, ci-contre, la désignation des plans anatomiques (figure 1.29).
Le reste de ce document utilisera ces désignations.

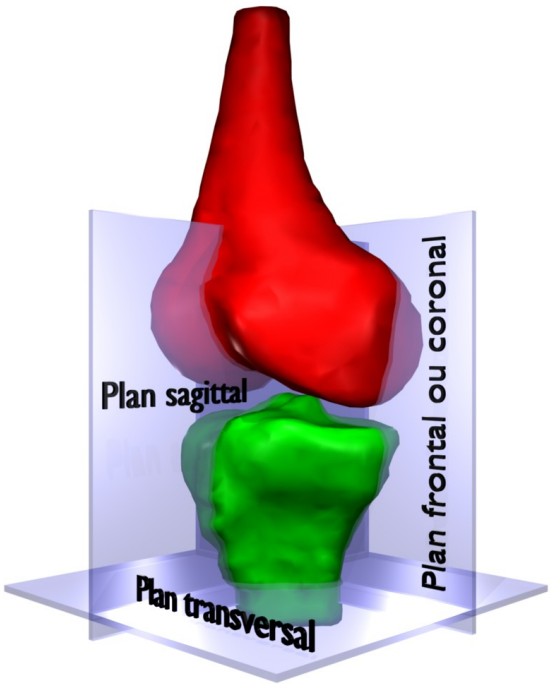


FIG. 1.29 – Plans anatomiques du genou

1.3.2 Les os

Le genou est constitué de quatre os : le fémur, le tibia, la patella (rotule), la fibula (péroné) (voir figure 1.30).

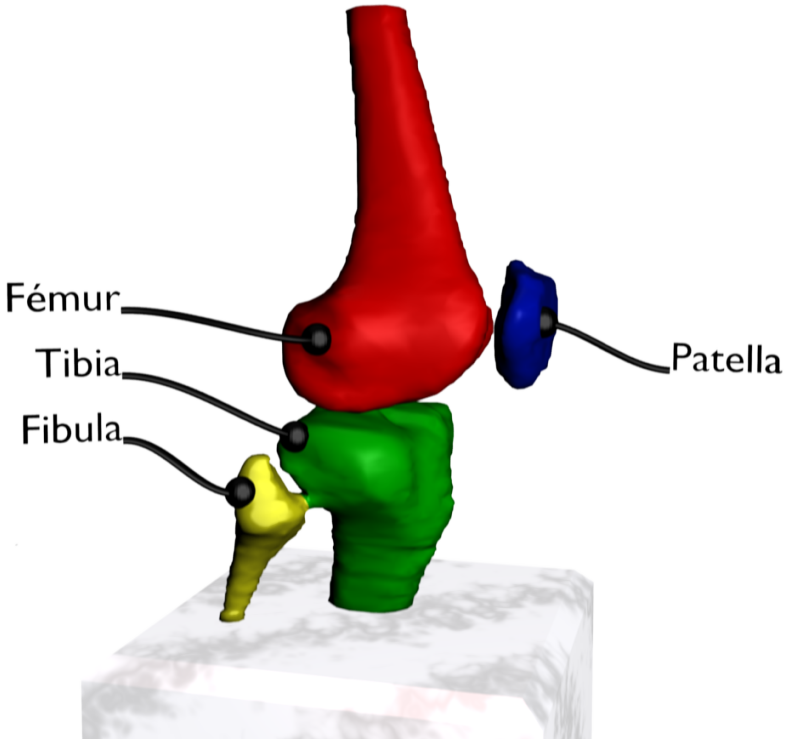


FIG. 1.30 – Les os du genou

1.3.3 Les surfaces en contact

Les surfaces en contact entre le tibia et le fémur les condyles et les glènes, illustrées ci-dessous en rouge et vert sur les figures 1.31 et 1.32.

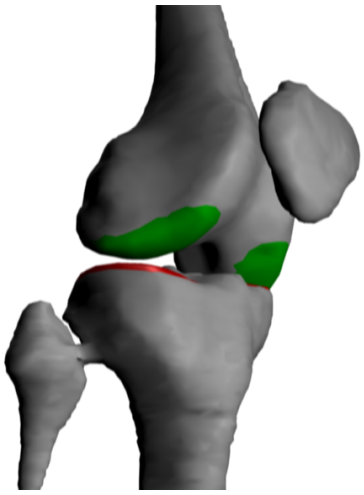


FIG. 1.31 – Les condyles (en vert) à l'extrémité du fémur

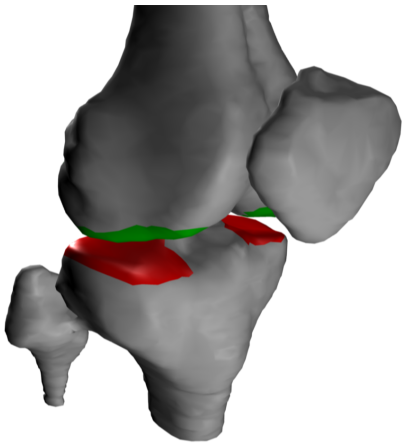


FIG. 1.32 – Les glènes (en rouge), sur le plateau tibial

1.3.4 Les ligaments

Les ligaments principaux du genou sont les suivants (figure 1.33) :

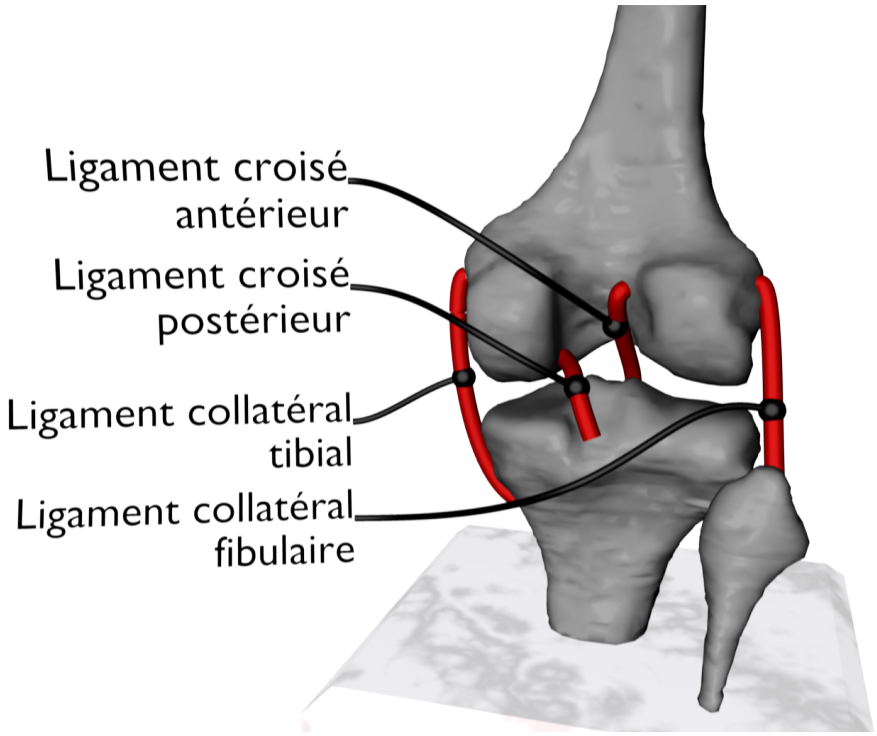


FIG. 1.33 – Les ligaments principaux du genou (genou vu de dos)

Le ligament collatéral tibial est aussi parfois appellé "Ligament latéral interne" (LLI) et le ligament collatéral fibulaire, "Ligament latéral externe" (LLE).

Chaque ligament travaille uniquement en traction.

1.3.5 Fonctionnement du genou

Complexité du système

Le genou est un organe très complexe, comme on peut le constater sur les vues 1.34 et 1.35.

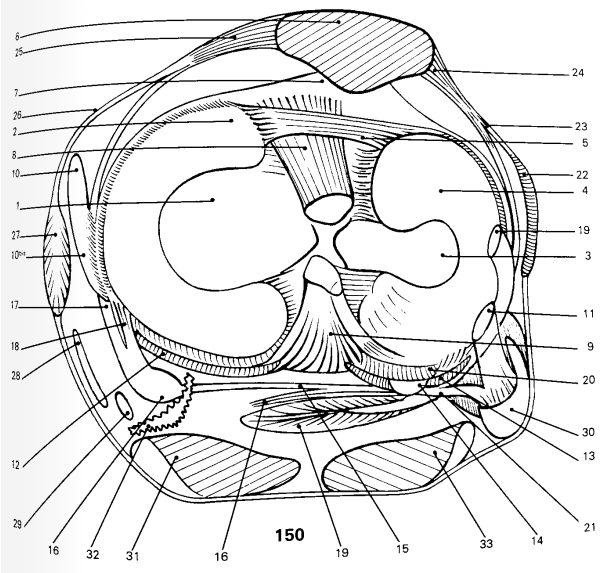


FIG. 1.34 – Coupe transversale issue du Kapandji [1], pour exemple

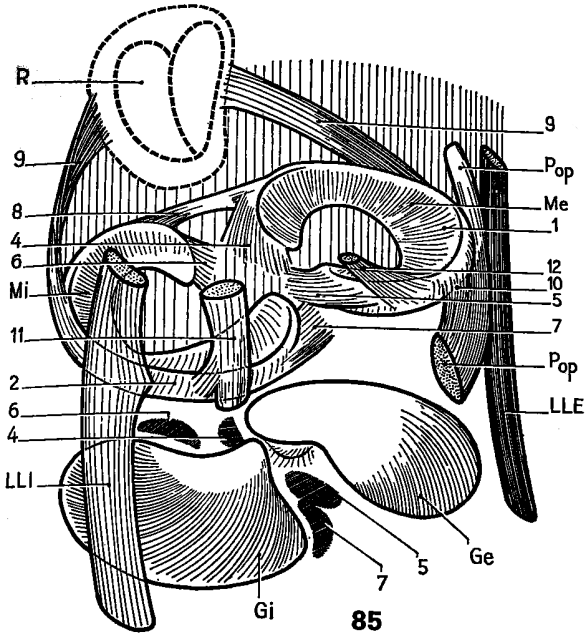


FIG. 1.35 – Vue 3D issue du Kapandji [1], pour exemple

Devant la complexité de ce mécanisme, nous nous limitons pour notre modèle, dans un premier temps, aux éléments essentiels du genou.

Simplification

D'après le livre de référence sur le genou ("Kapandji"[1]), les ménisques sont libres sur le plateau tibial et sont chassés par le tibia et le fémur ("comme un noyau de cerise entre deux doigts"). Leur rôle est de répartir la pression sur le tibia et le genou peut fonctionner sans eux.

La patella (rotule) sert principalement à transmettre l'effort musculaire du quadriceps au tibia pour déplier la jambe.

Les éléments les plus importants du genou, au niveau mécanique et dans le cadre d'un fonctionnement passif (sans tonus musculaire), sont donc : le tibia, le fémur et les ligaments croisés.

Ce fonctionnement de base du genou est illustré par les maquettes suivantes.

Maquettes du livre "Physiologie Articulaire, membre inférieur" [1]

A la fin du livre A.I. Kapandji propose les maquettes suivantes (à réaliser sur papier épais) (voir figures 1.36 à 1.39) pour bien comprendre le fonctionnement des ligaments croisés. D'après ce livre de référence, les ligaments croisés peuvent êtres vus comme un mécanisme à 4 barres. Le plateau supérieur décrit une trajectoire fixe par rapport au plateau inférieur.

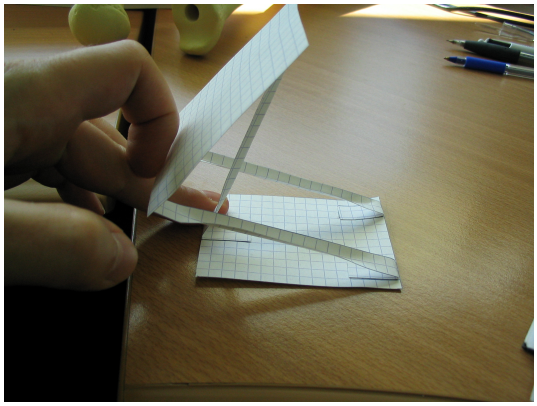


FIG. 1.36 – Mécanisme à 4 barres

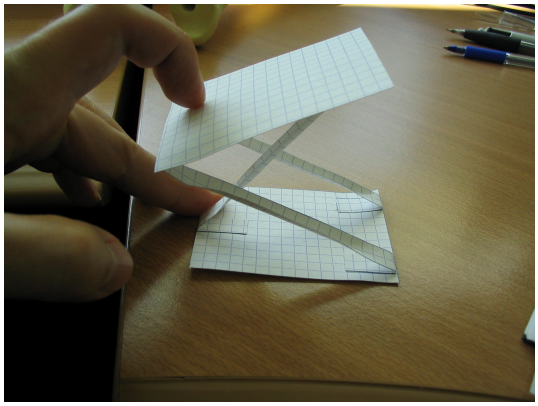


FIG. 1.37 – Mécanisme à 4 barres

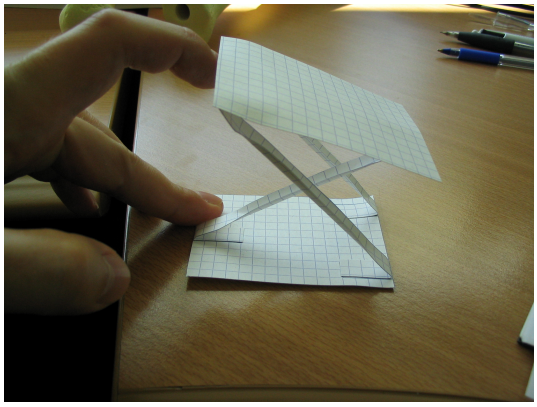


FIG. 1.38 – Mécanisme à 4 barres

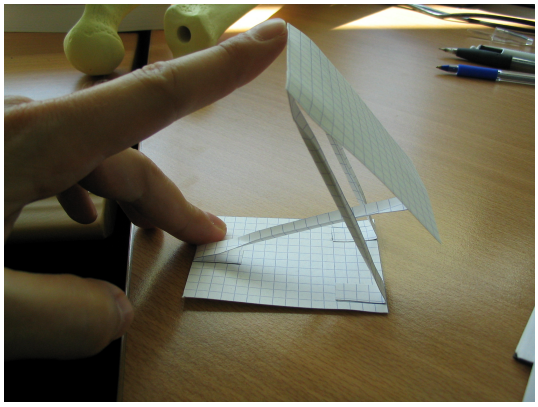


FIG. 1.39 – Mécanisme à 4 barres

Avec la maquette suivante, on vérifie que la trajectoire du plateau tibial et de la rotule décrivent l'enveloppe du fémur (voir figures 1.40 et 1.41).

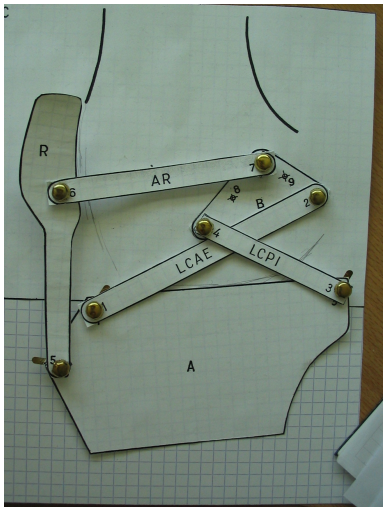


FIG. 1.40 – Enveloppe formée par la rotule et le plateau tibial

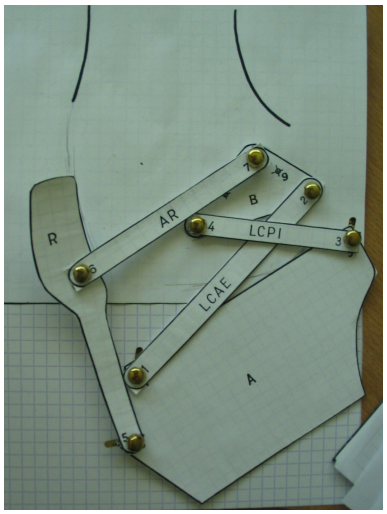


FIG. 1.41 – Enveloppe formée par la rotule et le plateau tibial

1.3.6 Nature du contact entre le tibia et le fémur

Dans les cas de flexion faible, le fémur roule sans glissement sur le tibia. Par contre, plus le genou fléchit, plus il se produit de glissement entre ces deux os.

1.3.7 Limitations en rotation dans le plan transversal

La limitation angulaire du fémur par rapport au tibia dans le plan transversal, dans le sens trigonométrique (flèche noire sur les figures 1.42 et 1.43) est assurée par les ligaments collatéraux.

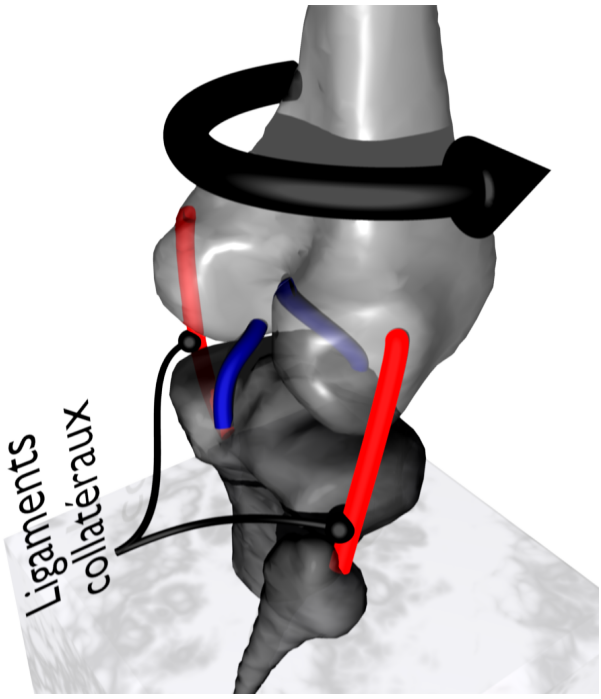


FIG. 1.42 – Ligament collatéraux, vue globale

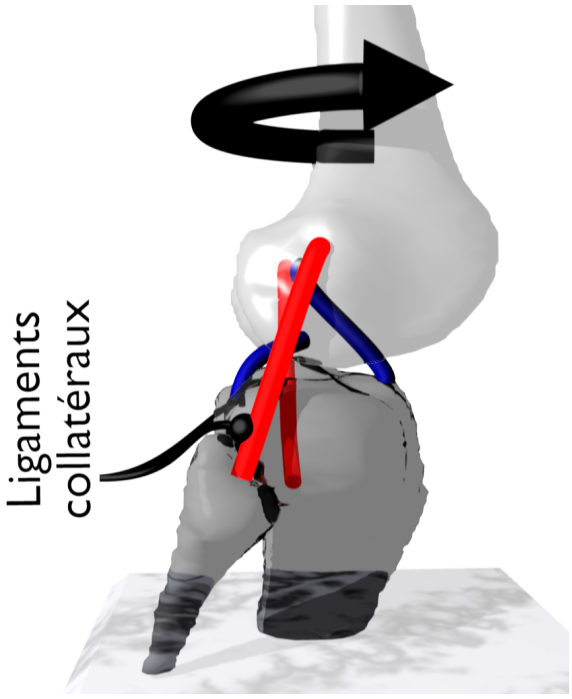


FIG. 1.43 – Ligament collatéraux, vue latérale

En effet, on remarque sur la figure 1.43, avec l'effet de transparence des os, que les ligaments collatéraux sont croisés dans le sens opposé à celui des ligaments croisés.

Inversement, dans le sens horaire, ce sont les ligaments croisés qui assurent la limitation angulaire (voir figures 1.44 et1.45).

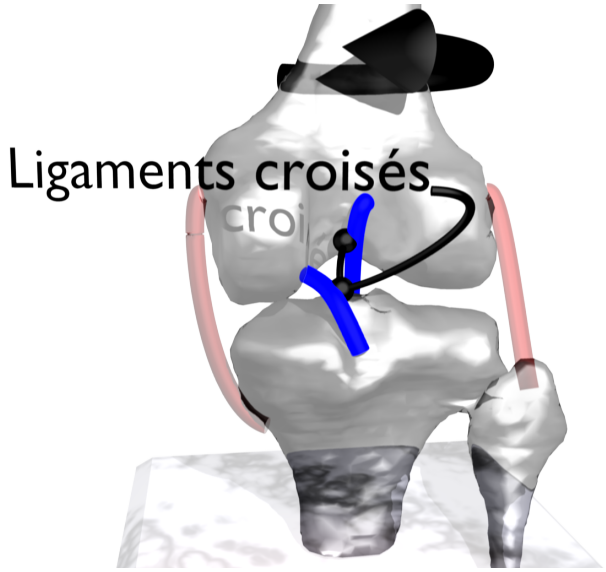


FIG. 1.44 – Ligament croisés, vue globale

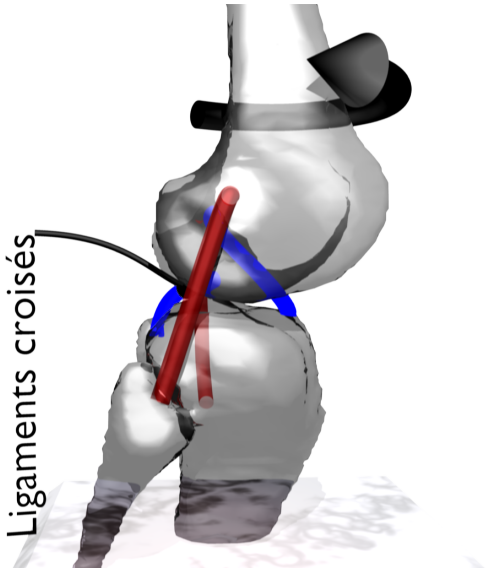


FIG. 1.45 – Ligament croisés, vue latérale

Chapitre 2

Réalisations

2.1 Déroulement chronologique du stage

Au cours des six mois de stage, la maquette de simulation du genou a évolué vers un outil généraliste pour préparer des scènes médicales à simuler avec SOFA.

Les sections ci-après montrent rapidement l'évolution de l'architecture du logiciel et l'évolution des maillages utilisés pour le modèle, au cours du stage, dans l'ordre chronologique. Chaque partie significative est détaillée dans des parties annexes auxquelles je ferais référence dans la suite de ce chapitre.

2.1.1 Choix de la modélisation

Dans un premier temps, nous utilisons une gestion des collisions entre solides appelée "méthode des pénalités" (ressorts et amortisseurs) et nous modélisons les ligaments par un système de masses-ressorts.

Sur les figures 2.1 et 2.2 ci-dessous, on observe en bleu les sommets en "surveillance" de contact et ceux qui sont déjà en contact (en magenta). Les solides se repoussent sans s'interpénétrer. Les chapelets de masses-ressorts sur les côtés représenterons les ligaments (figure 2.1 uniquement).

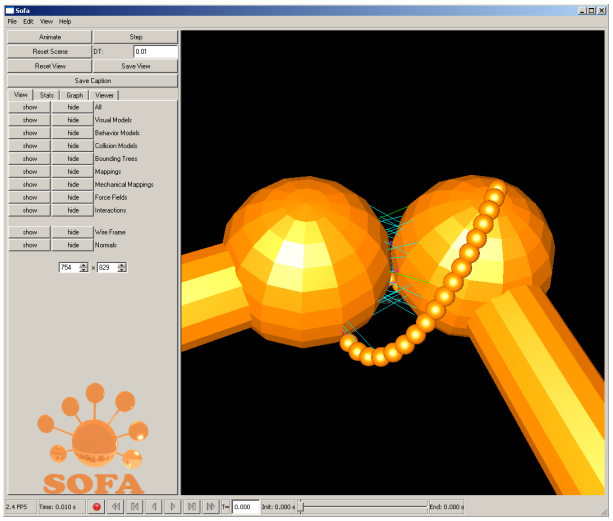


FIG. 2.1 – Représentation des forces d'interaction dans SOFA

FIG. 2.2 – Représentation des forces d'interaction dans SOFA

La distance de détection des contacts est paramétrable dans SOFA. Elle permet de simuler l'épaisseur des cartilages alors que nous utilisons les maillages d'os nus.

Le côté "interactif" par défaut de SOFA tient du fait que l'on peut à tout moment attacher un ressort sur un des sommets des maillages de la scène et le tirer à la souris pour simuler un effort ponctuel (il suffit d'enfoncer la touche SHIFT et de cliquer sur le sommet souhaité pour interagir avec la scène). Des développements en cours permettent d'utiliser des périphériques plus sophistiqués (à retour d'effort) pour interagir avec les scènes SOFA.

2.1.2 Mode opératoire

L'objectif consiste, dans un premier temps, à réaliser une maquette "passive" (sans tonus musculaire) avec un couple tibia-fémur alignés en extension horizontale. Le fémur est fixe et le tibia est soumis à la gravité. Lors de son basculement, le tibia doit effectuer une légère rotation axiale, d'après le D^{r} Palombi. Si l'on inverse la gravité, le genou ne doit pas fléchir dans le sens inverse.

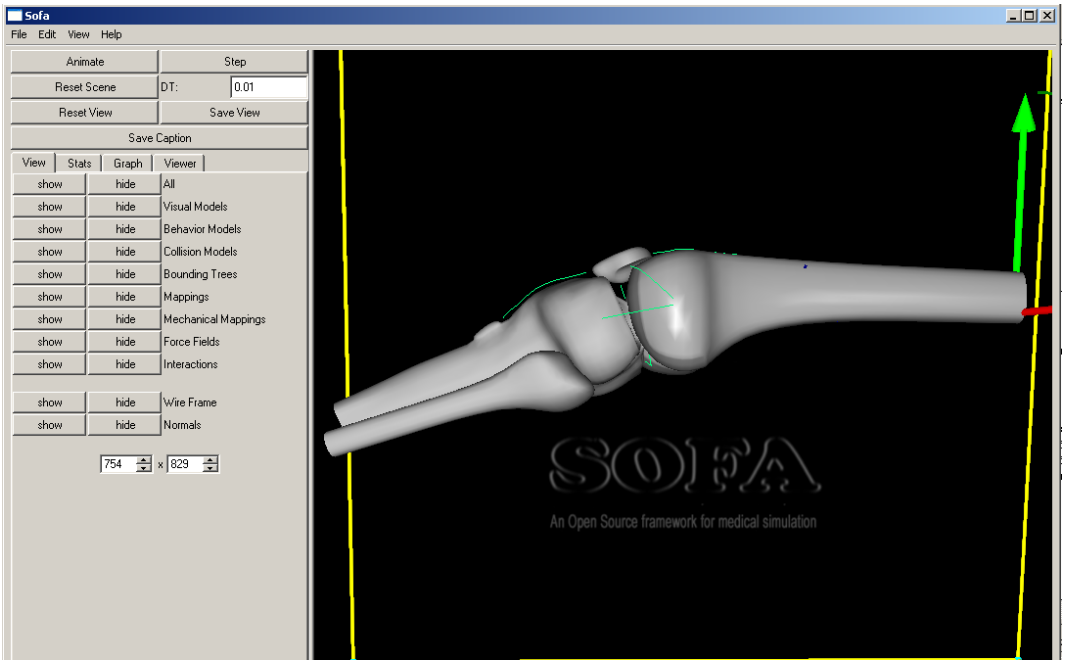


FIG. 2.3 – Mode opératoire : un fémur fixe et le tibia libre, soumis à la gravité

2.1.3 Pour commencer : une scène C++ avec des paramètres en dur

Il existe deux façons de produire des scènes SOFA, soit avec un fichier XML (au format "SOFA"), soit avec un programme C++ qui utilise SOFA. Dans mon cas, j'avais besoin de calculer les positions d'initialisation des ligaments (constitués de particules), et j'avais déjà des exemples de scènes en C++, j'ai donc commencé à construire un arbre de scène SOFA en C++.

Cette première version du logiciel (voir schéma figure 2.5) a donc consisté à construire l'arbre de scène en C++. Le listing 2.1 illustre le principe de construction de cet arbre, l'initialisation de la scène SOFA (ligne 26) et l'appel de la boucle de calcul de SOFA, ligne 29.

Ce type de fonctionnement m'a permis de construire une première scène composée de deux rotules (voir figure 2.4). Le détail de cette scène est détaillé en annexe, dans la section 4.2, page 49.

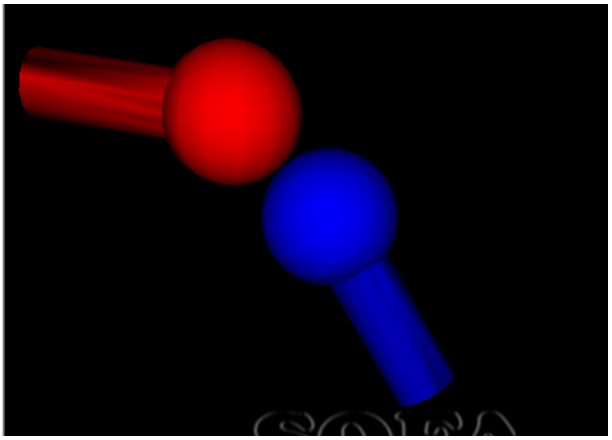


FIG. 2.4 – Première scène construite avec SOFA

Listing 2.1 – version 1 : construction d'une scène SOFA en C++

```
#include <sofa/core/...
...
#include <sofa/component/...
...
#include <sofa/simulation/tree/Simulation.h>

int main(int argc, char** argv) {
    sofa::helper::parse("This is a SOFA application.")
        (argc,argv);

    sofa::gui::SofaGUI::Init( argv[0] );

    // Racine du graphe de scène
    sofa::simulation::tree::GNode* node_root = new sofa::simulation::tree::GNode;
    node_root->setName( "root" );

    GNode * node_scene = new GNode;
    ...
    node_root->addChild( node_scene );
    ...
    sofa::component::odesolver::CGImplicitSolver* solver = new sofa::component::
        odesolver::CGImplicitSolver;
    ...
    node->addObject( solver );
    ...
    // Init the scene
    sofa::simulation::tree::getSimulation()->init(node_root);
    ...
    // Run the main loop
    sofa::gui::SofaGUI::MainLoop(node_root);
```

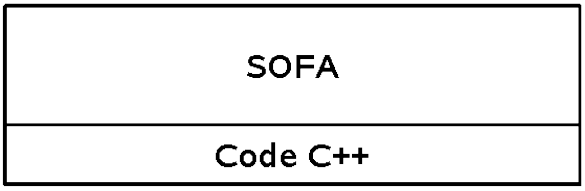


FIG. 2.5 – La première version du logiciel pour réaliser la scène du genou avec SOFA : du code C++ compilé avec SOFA avec des paramètres en dur

Suite à ce premier essai, j'ai commencé à utiliser le même programme avec les maillages fournis par François Boux de Casson (voir figures 2.6 et 2.7).

FIG. 2.6 – Premier maillage de tibia (gauche) fourni par François Boux de Casson

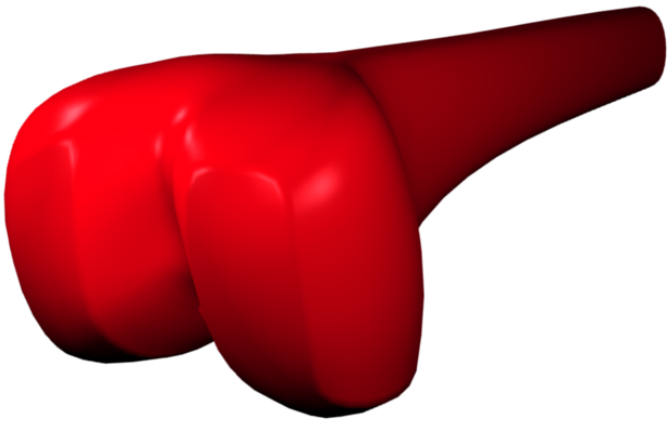


FIG. 2.7 – Premier maillage de fémur (gauche) fourni par François Boux de Casson

Pour calculer le centre de gravité de ces maillages et leurs éléments d'inertie, j'ai utilisé le code C (Volint.C) de Brian Mirtich qui fonctionne avec le format "Polyhedron". Sofa utilise le format de maillage 3D "obj (WaveFront)". J'ai obtenu le format "Polyhedron" avec le programme "ObjToPolyhedron" (voir en annexe la liste des programmes C++ console écrits, section 4.1.5, page 47).

Ensuite, tous les sommets du maillage étaient translatés du vecteur $-\overrightarrow{OG}$ pour que le zéro du maillage soit situé sur le centre de gravité. Cette translation était effectuée à l'aide du programme console "TranslateObj" (voir section 4.1.5, page 47).

Les ligaments ont été modélisés avec des chapelets de masses-ressorts. Le détail de cette modélisation est détaillée en annexe, section 4.8, page 81.

Pour régler les paramètres de la scène, comme la position des os, la position des ligaments, leur raideur,leur amortissement, il fallait modifier le code (voir exemples en annexe listing 4.7 page 81 et 4.8 page 82).

Devant les nombreux réglages à effectuer et les compilations associées, j'ai décidé d'exporter toutes les informations nécessaires à la modélisation dans un fichier XML externe.

2.1.4 Amélioration : scène C++ avec paramètres XML

La figure 2.8 illustre l'amélioration du code précédent. On peut modifier les paramètres de la scène sans avoir à recompiler et quand même bénéficier du langage C++ pour positionner les particules des ligaments sur la courbe voulue.

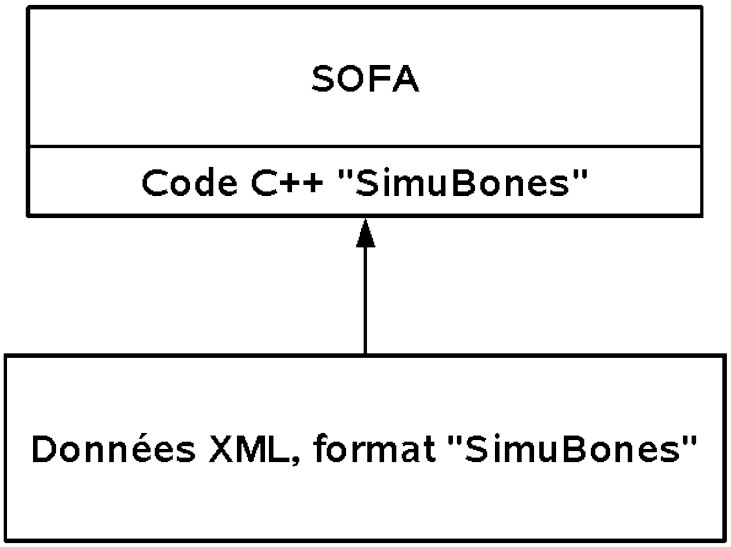


FIG. 2.8 – Première amélioration du programme précédent avec l'utilisation d'un fichier XML pour configurer tous les paramètres de la scène

Le fichier XML contient :
- La liste des maillages à utiliser dans la scène (les os)
- Pour chaque os est défini un liste d'"attaches" nommées. Une attache est composée d'un point et d'un vecteur directeur
- Une liste de ligaments, avec pour chaque ligament :
  - Le nombre de sphères (modèle réalisé par un chapelet de masses-ressorts)
  - Sa masse
  - Sa longeur initiale
  - La valeur de l'amortisseur et de la raideur des ressorts
  - L'attache source (exemple : "Cone bleu - attache A", figure 2.9)
  - L'attache destination (exemple : "Cylindre vert - attache C", figure 2.9)

Le choix d'associer à chaque point d'attache un vecteur directeur vient du besoin d'initialiser les ligaments dans une position qui n'interpénètre pas d'autres éléments de la scène. On voit sur les figures 2.10, 2.11 et 2.12 que l'initialisation du ligament représenté ne pourrait pas s'effectuer sur une ligne droite sans pénétrer dans les os.

Le principe des attaches se résume donc à la figure ci-dessous (figure 2.9).

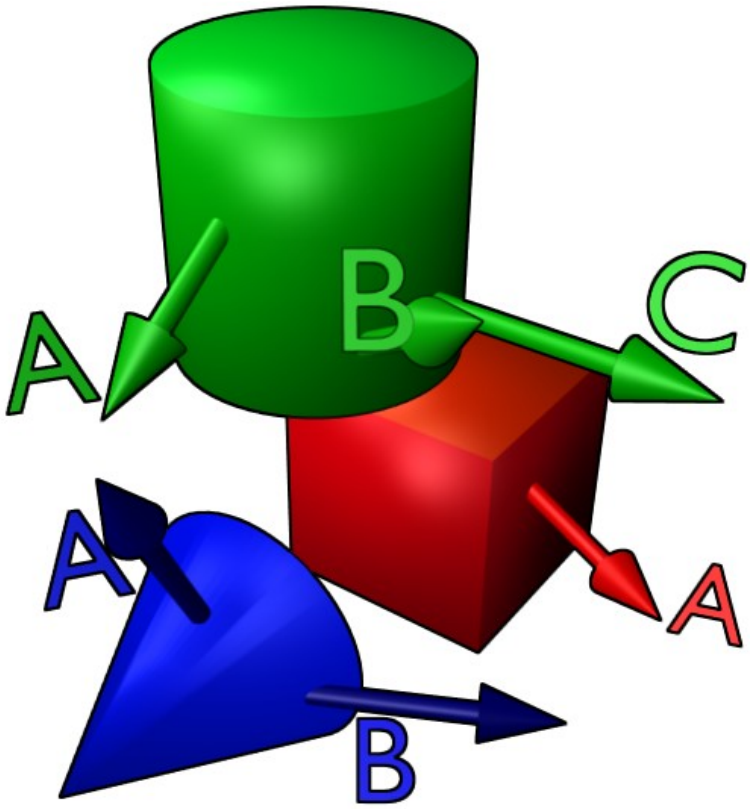


FIG. 2.9 – Principe des “attaches” sur les os : un point et un vecteur directeur

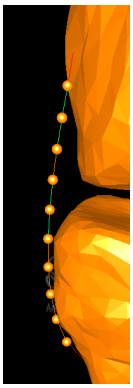


FIG. 2.10 – Position initiale d’un ligament latéral

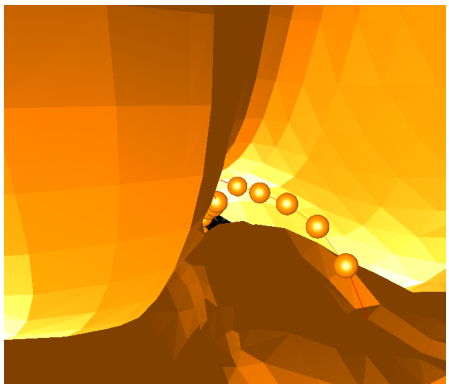


FIG. 2.11 – Position initiale d’un ligament croisé entre le fémur et le tibia

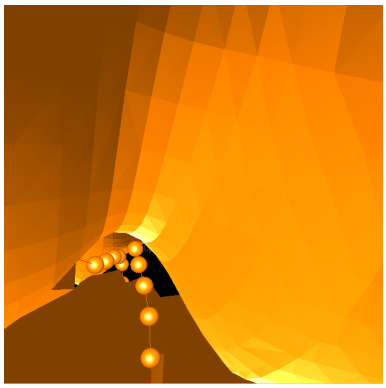


FIG. 2.12 – Position initiale d’un ligament croisé entre le fémur et le tibia

Après de nombreux essais il s’est révélé très difficile et fastidieux de déplacer les os, les points d’attache et les vecteurs directeurs directement par leurs coordonnées dans le fichiers XML. J’ai donc cherché à améliorer le système en utilisant un outil graphique pour modifier la position des éléments à la souris.

Le projet SOFA est basé sur du logiciel libre et il n’existe apparemment qu’un seul logiciel 3D libre personnalisable avec des scripts, c’est Blender.

2.1.5 Amélioration : scène C++ avec paramètres XML et configuration avec Blender

J'ai donc personnalisé Blender (avec un script Python relié au menu "File / Exporter vers un fichier XML SimuBones") pour mettre à jour la position des objets dans mon fichier XML. Ce nouveau principe de fonctionnement est illustré par la figure 2.13.

Les objets de type "os" sont représentés par des objets de type "Mesh" dans Blender et les ligaments par des objets de type "Curve". Les ligaments dans Blender sont des courbes de Bézier 3D.

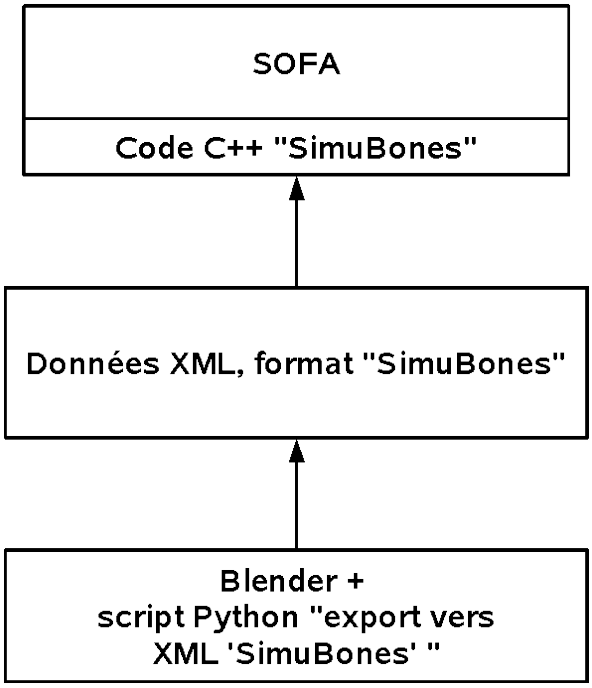


FIG. 2.13 – Facilitation de la modélisation des éléments de la scène avec Blender

On voit, figure 2.14, une scène composée de deux os simplifiés et de deux ligaments. À cette scène est associée le fichier XML de configuration. Le lien entre la scène Blender et le fichier XML est le nom des objets (cela impose d'avoir unicité des noms dans la scène).

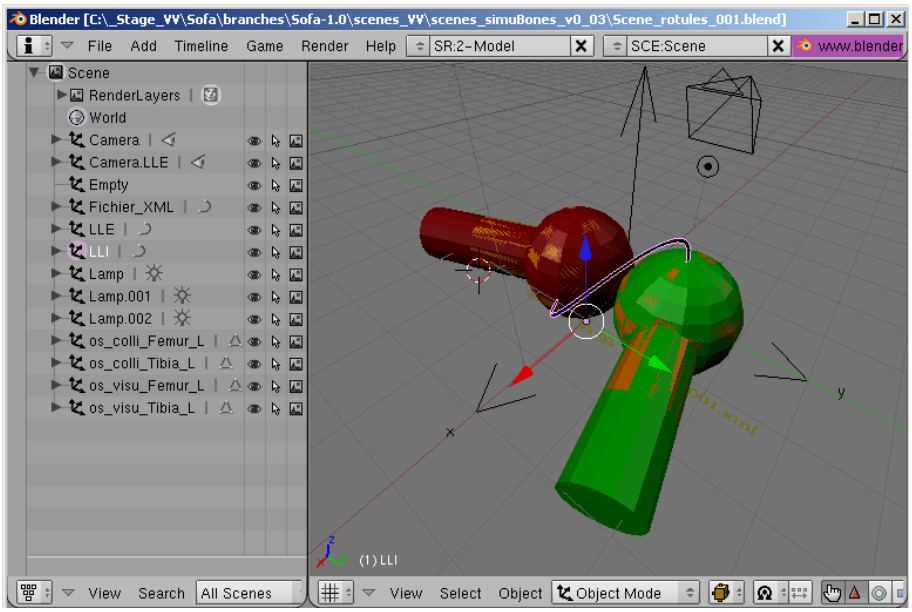


FIG. 2.14 – Configuration d'une scène SimuBones dans Blender

Grâce à ce système on obtient dans SOFA exactement la même scène que celle modélisée dans Blender (voir figures 2.15 à 2.18) :

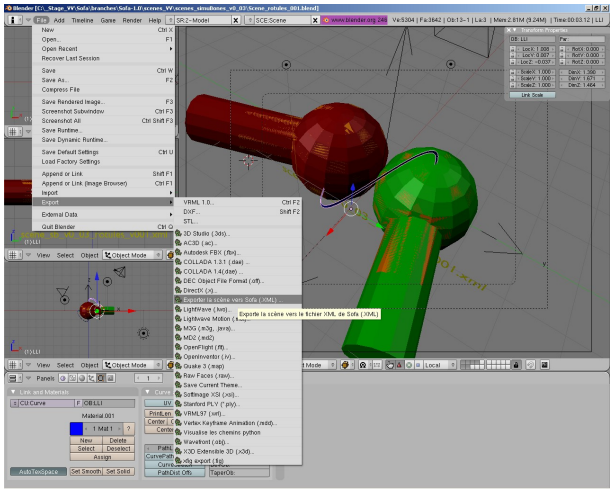


FIG. 2.15 – Export de la scène depuis Blender vers le fichier XML de configuration

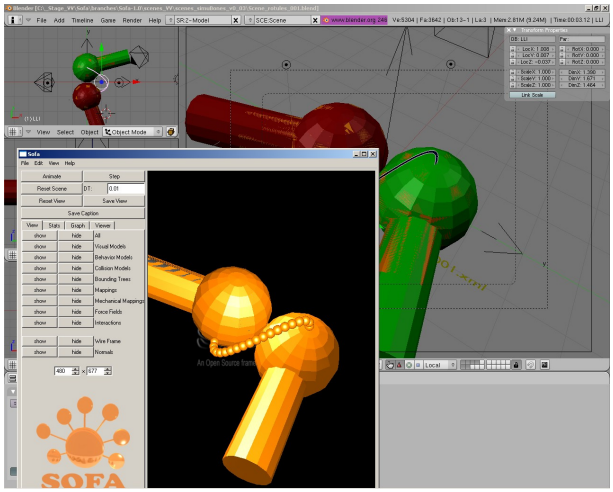


FIG. 2.16 – Import dans SOFA

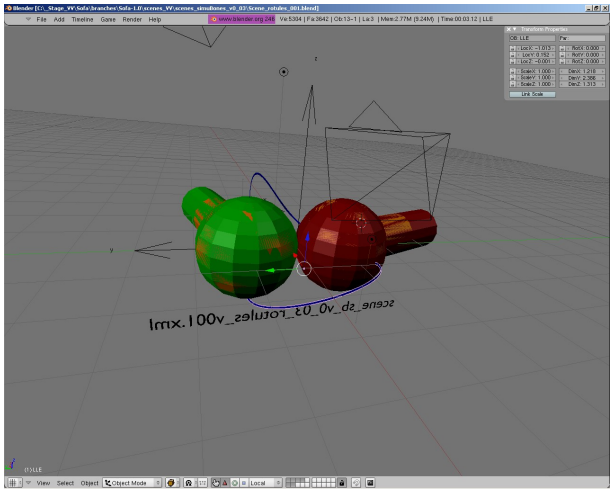


FIG. 2.17 – vue de la scène dans Blender

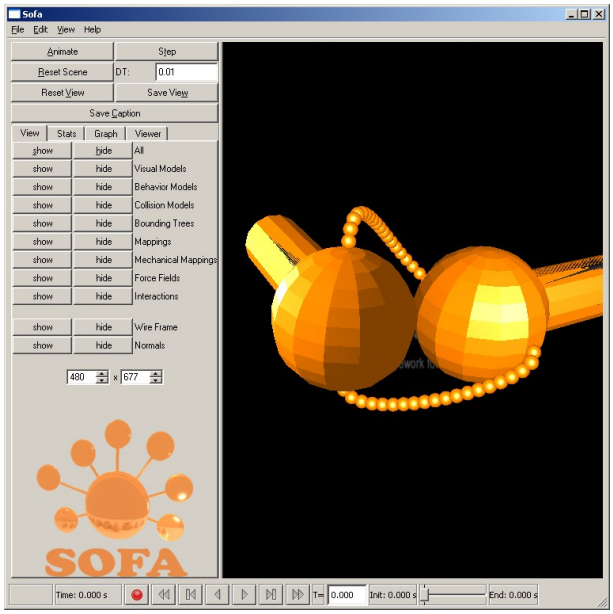


FIG. 2.18 – Import dans SOFA, on remarque que les courbes de Bézier ont été remplacés par des chapelets de masses-ressorts

Après avoir validé l'utilisation de Blender pour placer les objets de la scène, j'ai écrit un mini tutoriel Blender (de 5 pages) pour configurer l'environnement de Blender et déplacer les objets. Il se trouve en annexe, section 4.12, page 90. J'en ai écrit un autre plus tard pour écrire des scripts Python dans Blender, toujours en annexe, section 4.13, page 95. Comme ces tutoriels n'ont pas beaucoup de chance d'être lus, perdus dans ce gros document et qu'ils n'ont rien de spécifique à ce projet, j'ai placé les dernières versions sur mon site internet, sur cette page http://www.vincentvansuyt.com/bld_Tutoriels.htm.

Le logiciel "SimuBones" est détaillé en annexe, section 4.3, page 51, principalement au niveau du format du fichier XML utilisé.

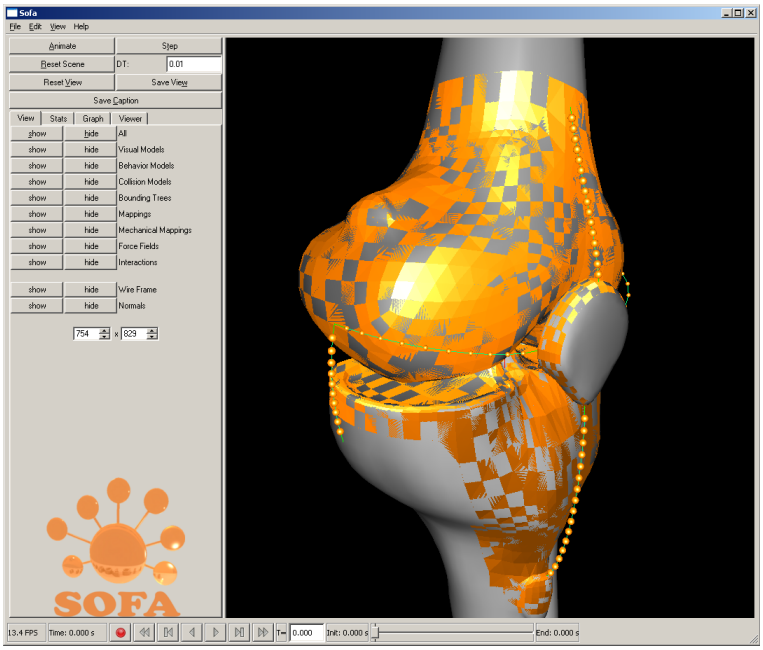


FIG. 2.19 – Exemple de positionnement du maillage surfacique, du maillage de collision (en orange) et placement initial des ligaments dans SOFA

Il ne reste plus qu'à appuyer sur le bouton "Animate" de SOFA pour que la scène prenne vie (figure 2.20)

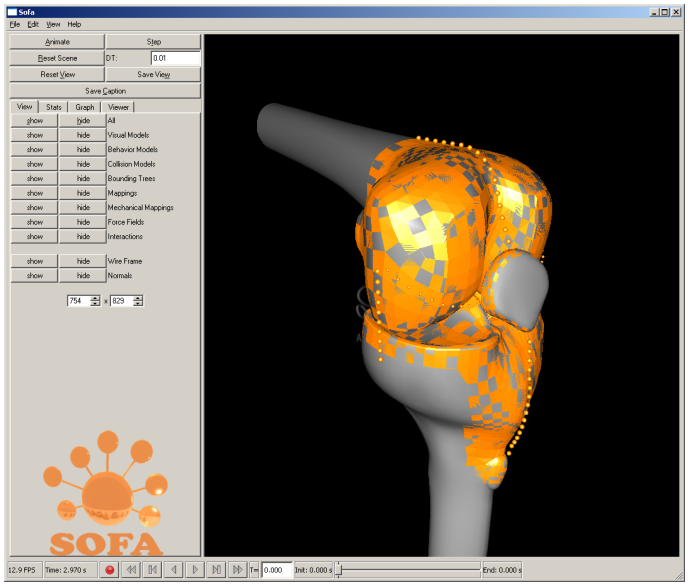


FIG. 2.20 – Animation de la scène avec SOFA

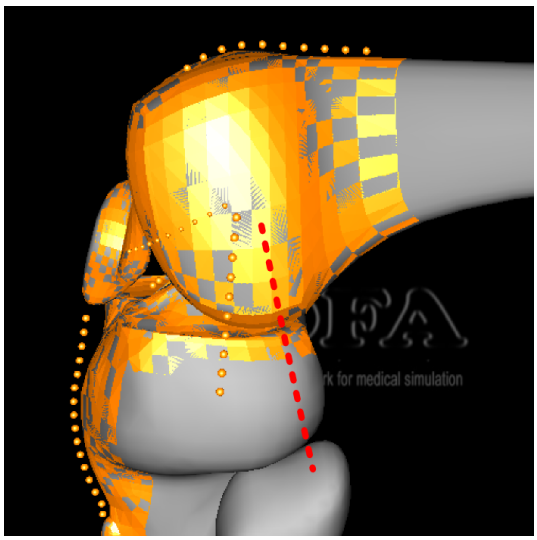


FIG. 2.21 – Erreur de positionnement ligament

Malheureusement, cette configuration et cette géométrie des os n'ont pas permis d'obtenir de résultat convaincant. Lorsque les ligaments sont attachés sur les parties adéquates de chaque os (le ligament latéral externe devrait être sur la position tracée en pointillés sur la figure 2.21), le fémur finit par sortir du plateau tibial vers l'arrière. Nous nous sommes aperçu plus tard (lire section 2.1.8, page 30) que la géométrie des os fournis par François Boux de Casson ne correspondait pas à une géométrie réaliste. D'après le livre "Physiologie articulaire" d'A.I. Kapandji [1] et ses maquettes (voir figures 1.40 et 1.41 page 16), la géométrie du genou est subtile et intimement lié à la position des ligaments.

Les deux paragraphes suivants sortent un peu du fil directeur. Pour la suite de la réalisation d'un modèle de genou, il faut passer directement à la section 2.1.8, page 30 pour la suite de l'évolution du modèle.

2.1.6 Généralisation de l'utilisation de Blender pour toutes les scènes SOFA

Suite à cette découverte des possibilités de Blender, j'ai appliqué le même principe aux scènes SOFA qui ne disposaient d'aucun éditeur de position graphique (avec l'autorisation de François Faure). La différence avec la version précédente est que les scènes SOFA XML sont importées et exportées de Blender, et que la configuration de l'arborescence d'une scène SOFA peut être quelconque.

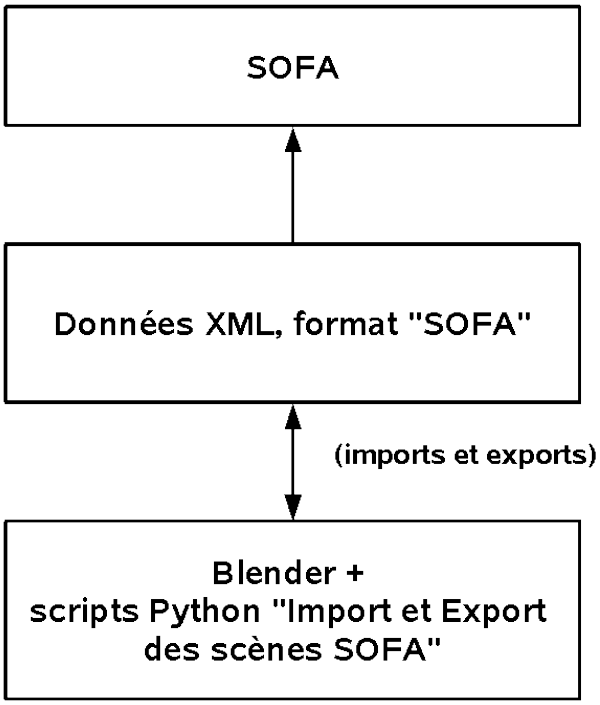


FIG. 2.22 – Modification des scènes SOFA avec Blender

Voici, figures 2.23 à 2.28, un exemple d'utilisation des scripts d'import-export des scènes SOFA pour Blender.

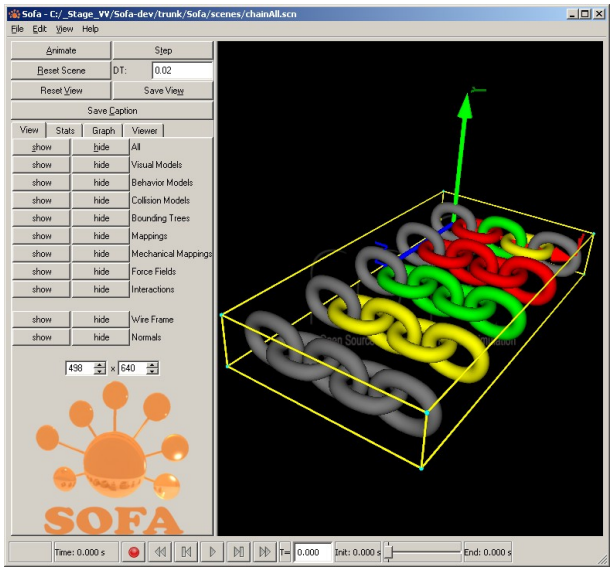


FIG. 2.23 – Scène de démonstration livrée avec SOFA ("ChainAll")

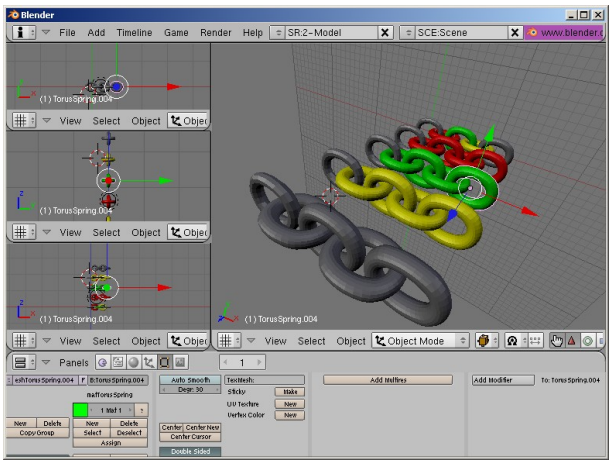


FIG. 2.24 – La même scène, importée dans Blender

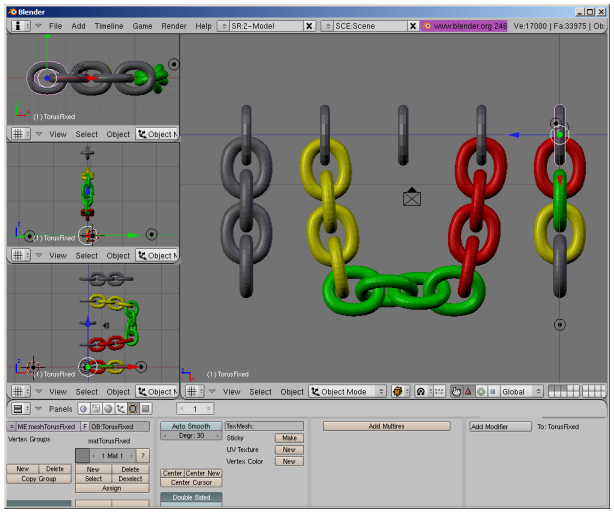


FIG. 2.25 – Modification de la scène avec Blender

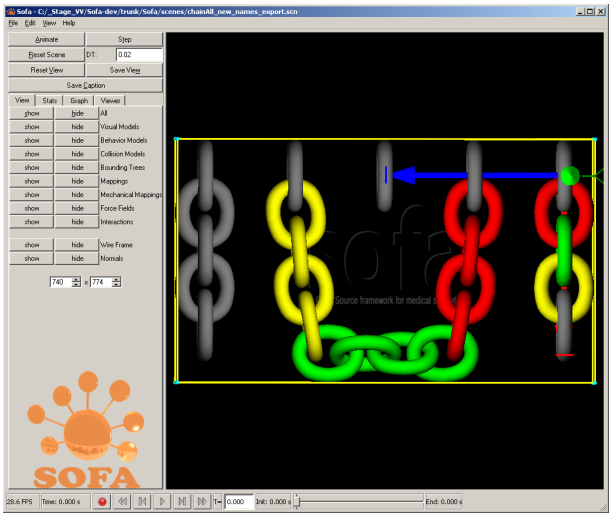


FIG. 2.26 – La scène dans SOFA après export de Blender

Dans la version de SOFA du mois d'août (deux mois après la version ci-dessus), l'import-export des scènes fonctionne toujours, malgré les modifications de SOFA (voir figures 2.27 et 2.28).

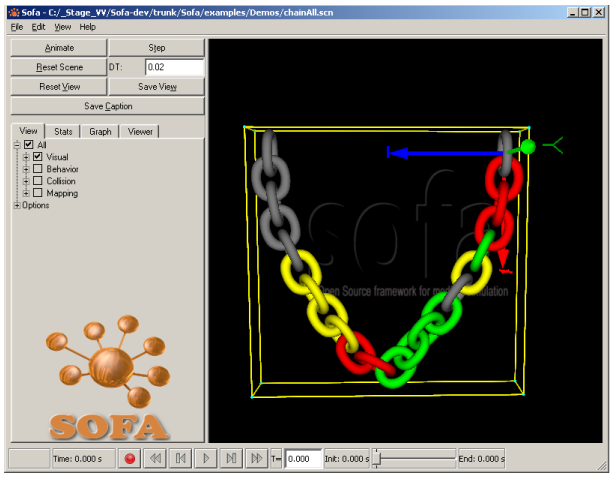


FIG. 2.27 – Scène livrée dans la version de SOFA du mois d'août

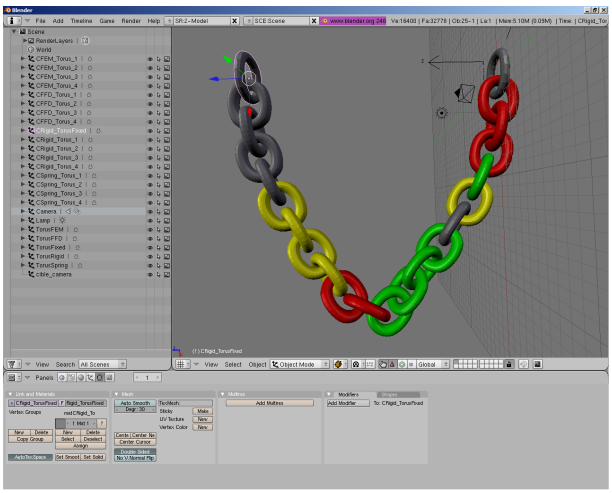


FIG. 2.28 – La scène SOFA dernière version importée dans Blender

Les imports-exports de Blender proposent des options lors de l'export. Les maillages peuvent être mis à jour avec Blender et être exportés ou non. Cette partie est détaillée en annexe, section 4.4, page 57.

2.1.7 Calcul des éléments d'inertie avec Blender

Les possibilités offertes par Blender m'ont poussé à améliorer la procédure de calcul des éléments d'inertie et de centre de gravité des maillages.

Alors qu'au début j'utilisais plusieurs programmes console C/C++ ("ObjToPolyhedron", "VolInt" [le programme C de Brian Mirtich] et "Translate Obj"), j'ai simplifié cette procédure en portant le programme C de Brian Mirtich en Python pour Blender. Ainsi, le code est utilisable sans avoir besoin de le compiler, il est disponible pour tous les formats de maillages proposés par Blender et il n'y a plus besoin de passer par une console et ligne de commande.

Ce programme est détaillé en annexe, section 4.6, page 74. Je l'ai aussi mis à disposition librement sur mon site internet avec le manuel d'utilisation, sur cette page : http://www.vincentvansuyt.com/bld_Scripts_maillages.htm. J'ai bien laissé en commentaire dans le code les termes d'utilisation de ce

programme tel que l'avait écrit Brian Mirtich (voir figure 2.29), et le fichier README a aussi été intégré en commentaires dans le script Python.

```
/*****************************************************
*                                                    *
*  volInt.c                                          *
*                                                    *
*  This code computes volume integrals needed for    *
*  determining mass properties of polyhedral bodies. *
*                                                    *
*  For more information, see the accompanying README *
*  file, and the paper                               *
*                                                    *
*  Brian Mirtich, "Fast and Accurate Computation of  *
*  Polyhedral Mass Properties," journal of graphics  *
*  tools, volume 1, number 1, 1996.                  *
*                                                    *
*  This source code is public domain, and may be used *
*  in any way, shape or form, free of charge.        *
*                                                    *
*  Copyright 1995 by Brian Mirtich                   *
*                                                    *
*  mirtich@cs.berkeley.edu                           *
*  http://www.cs.berkeley.edu/~mirtich               *
*                                                    *
*****************************************************/
```

FIG. 2.29 – Conditions d'utilisation du programme de Brian Mirtich

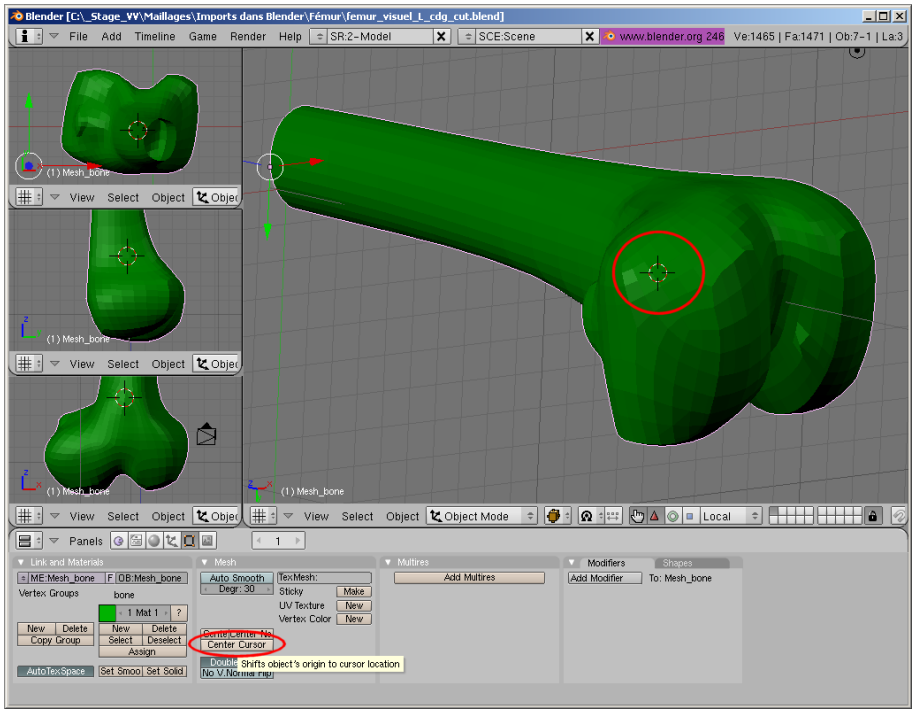


FIG. 2.30 – Exemple d'obtention du centre de gravité, avec Blender (voir en annexe pour les détails)

2.1.8 Amélioration de la géométrie du maillage des os du genou

Deux mois après le début du stage, nous avons reçu le livre de référence d'A.I. Kapandji sur le genou [1] dans lequel je me suis aperçu que le maillage initial des os que nous avons utilisé ne nous permettrait pas d'obtenir de bons résultats parce leur géométrie n'est pas réaliste, en particuler au niveau des glènes (comparer les profils des surfaces de contact des figures 2.32-47, 2.32-48 et 2.33).

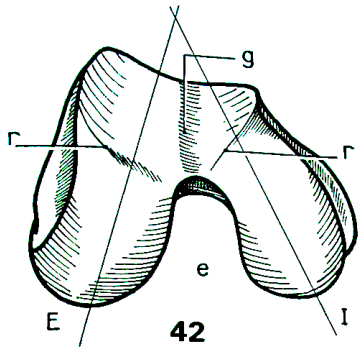


FIG. 2.31 – Plans de coupe de la figure 2.32

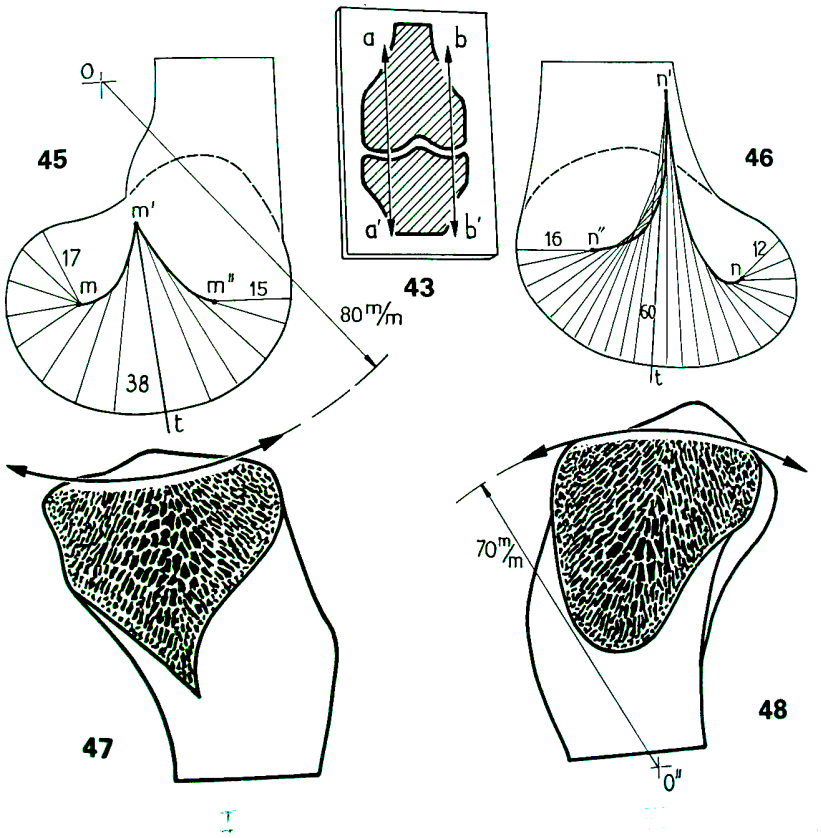


FIG. 2.32 – Illustations du profil théorique des glènes et condyles issue du Kapandji [1]

Heureusement, grâce à Olivier Palombi (le neurochirurgien et anatomiste du laboratoire), l'équipe du docteur Jean-Noël Ravey et Antonin Fontanile nous avons pu obtenir un rendez-vous pour passer un IRM sur le genou d'Antonin à l'hôpital sud de Grenoble.
Suite à cet examen nous avons récupéré des données volumiques au format DICOM qu'il a fallu segmenter pour en extraire les os. Le détail de cette opération est décrite en annexe, dans la section 4.5, page 65. N'étant pas radiologue, le résultat que j'ai obtenu lors de la segmentation n'est pas idéal. Cette segmentation devrait être améliorée plus tard par un spécialiste.

FIG. 2.33 – Premier maillage de tibia (gauche) fourni par François Boux de Casson

Voici (figures 2.34 et 2.35) le résultat obtenu à l'issue de la segmentation.

FIG. 2.34 – Genou IRM, vue gauche

FIG. 2.35 – Genou IRM, vue droite

On vérifie bien avec ce modèle les rayons de glènes annoncés par A.I. Kapanji (comparer les figures 2.36 et 2.37 avec les figures 2.32-47 et 2.32-48).

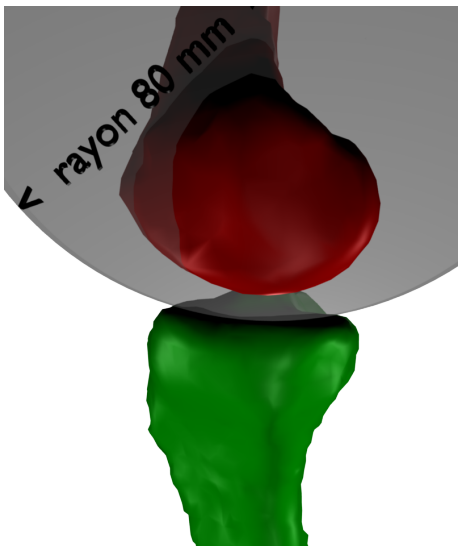


FIG. 2.36 – glène interne (rayon 80 mm)

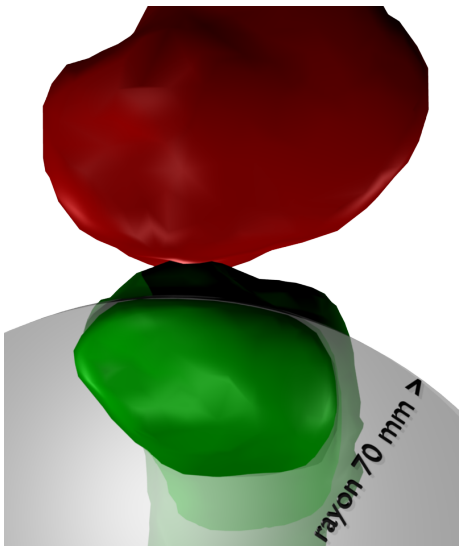


FIG. 2.37 – glène interne (rayon 70 mm)

Attention, ce nouveau modèle du genou provient d'une jambe droite, alors que le précédent modèle venait d'une jambe gauche.

2.1.9 Utilisation de la base de données “myCorporisFabrica” v0.1

Suite à l’obtention de ces précieux jeux de données sur le genou, l’idée est venue à François Faure et Olivier Palombi de centraliser toutes les données médicales dans une base de données anatomique, sur un serveur.

Le principe est de séparer la partie anatomique de la partie modélisation. La base “myCorporisCorpora” a été crée par Olivier Palombi au mois de Juillet. Ce nom a été inspiré par un ouvrage écrit par l’anatomiste André Vésale (1514 - 1564). Il est l’auteur d’un des célèbres livres sur l’Anatomie humaine, “De humani corporis fabrica”[1] (sur le fonctionnement du corps humain).

FIG. 2.38 – La couverture du livre “De humani corporis fabrica” d’André Vésale dont le titre a inspiré le nom de la base de données “myCorporisFabrica”

Avant “myCorporisFabrica” il existait un modèle de base de données anatomique nommée FMA (“Foundational Model of Anatomy”), mais elle ne correspondait pas à notre besoin.

La base de données “myCorporisFabrica” énumère tous les éléments du corps humain et leur attribue un identifiant “Entité Anatomique” et un nom. Chaque entité anatomique a un type (tendon, cartilage, ligament, os et organe).

Ensuite, les relations entre les entités anatomiques sont décrites dans les tables “a_sinseresur” et “a_faitpartiede”. Ce sont des tables à deux colonnes qui contiennent des numéros d’entité anatomiques.

En parallèle avec cette connaissance anatomique, la base “myCorporisFabrica” permet d’accueillir des données en provenance d’acquisitions médicales. Ces acquisitions s’insèrent dans la table “acquisition” et les éléments dont on a acquis les données s’insèrent dans la table “instances” avec leur position linéaire, angulaire, leur numéro d’acquisition, leur numéro d’entité anatomique et le fichier maillage associé.

La figure 2.39, page 33 représente le principe de fonctionnement de cette base de données.

La section 4.9, page 82, en annexe, contient le détail de la structure des tables de cette base et un exemple des données contenues.

[1] Ce livre est entièrement numérisé et disponible gratuitement sur le site de la bibliothèque universitaire de Tours (www.bvh.univ-tours.fr), ainsi que de nombreux autres ouvrages anciens.

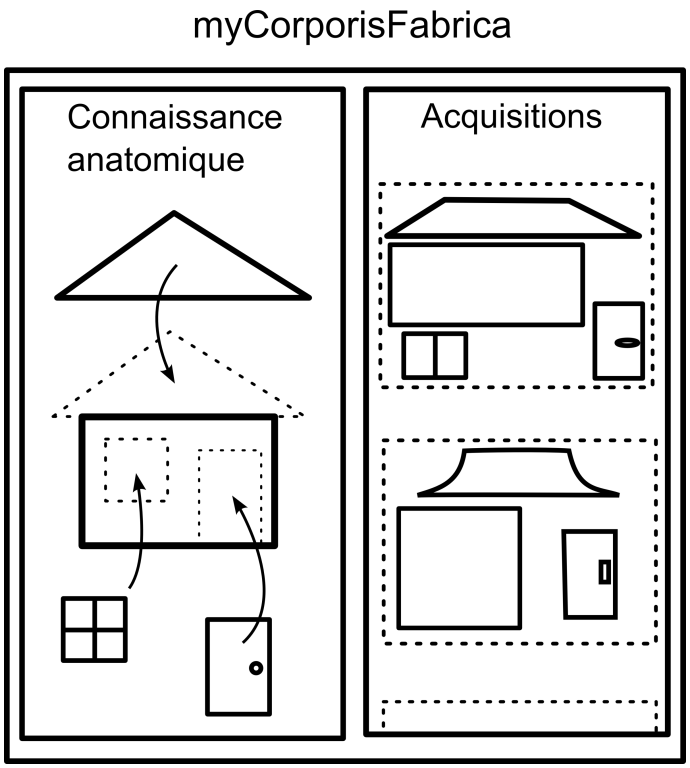


FIG. 2.39 – Principe de fonctionnement de la base

Le nouveau pipe-line de création de scènes SOFA à partir de données médicales devient celui dessiné figure 2.40 (cette partie sera détaillée plus loin). J'ai réalisé la partie "mise à jour de la base myCorporisFabrica" avec Blender (étape 2 de la figure 2.40) et le logiciel d'export des scènes SOFA à partir de la base de données (étape 3 de la figure 2.40).

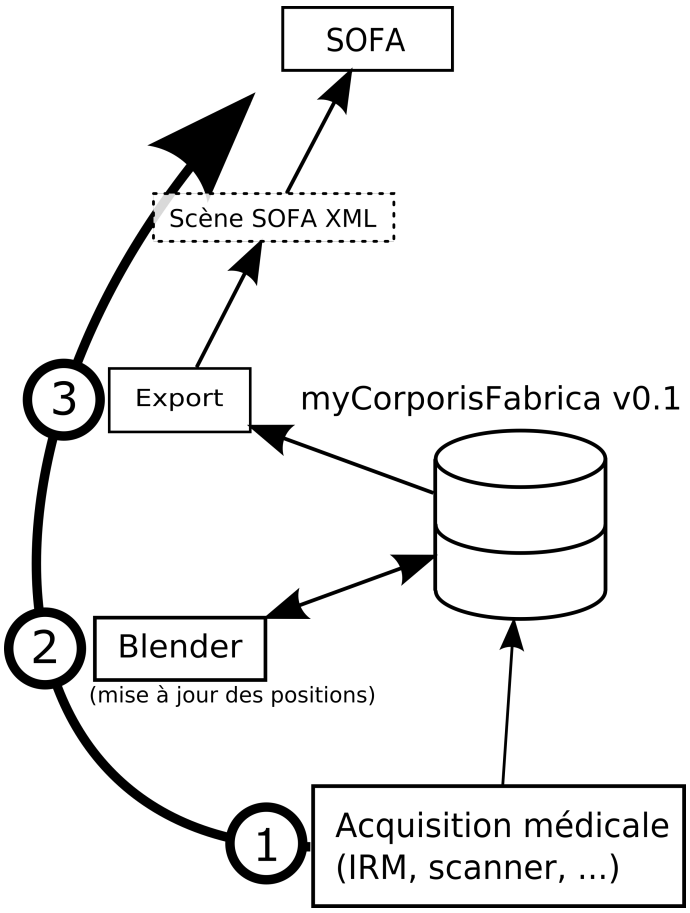


FIG. 2.40 – Plan d'ensemble de production de scènes médicales avec SOFA

2.1.10 Import des organes de myCorporisFabrica v0.1 dans Blender

Le but est d'importer un organe de la base de données pour mettre à jour la position des différents éléments. En prérequis il faut bien sûr avoir installé les librairies NumPy, SciPy, mySQL pour Python et que la base soit déjà alimentée avec des éléments. Pour l'instant cette partie est réalisée à partir de l'interface phpMyAdmin de EasyPhp qui héberge la base sur son serveur mySQL.

La procédure à suivre pour importer un organe est décrite par la liste des copies d'écran ci-dessous.

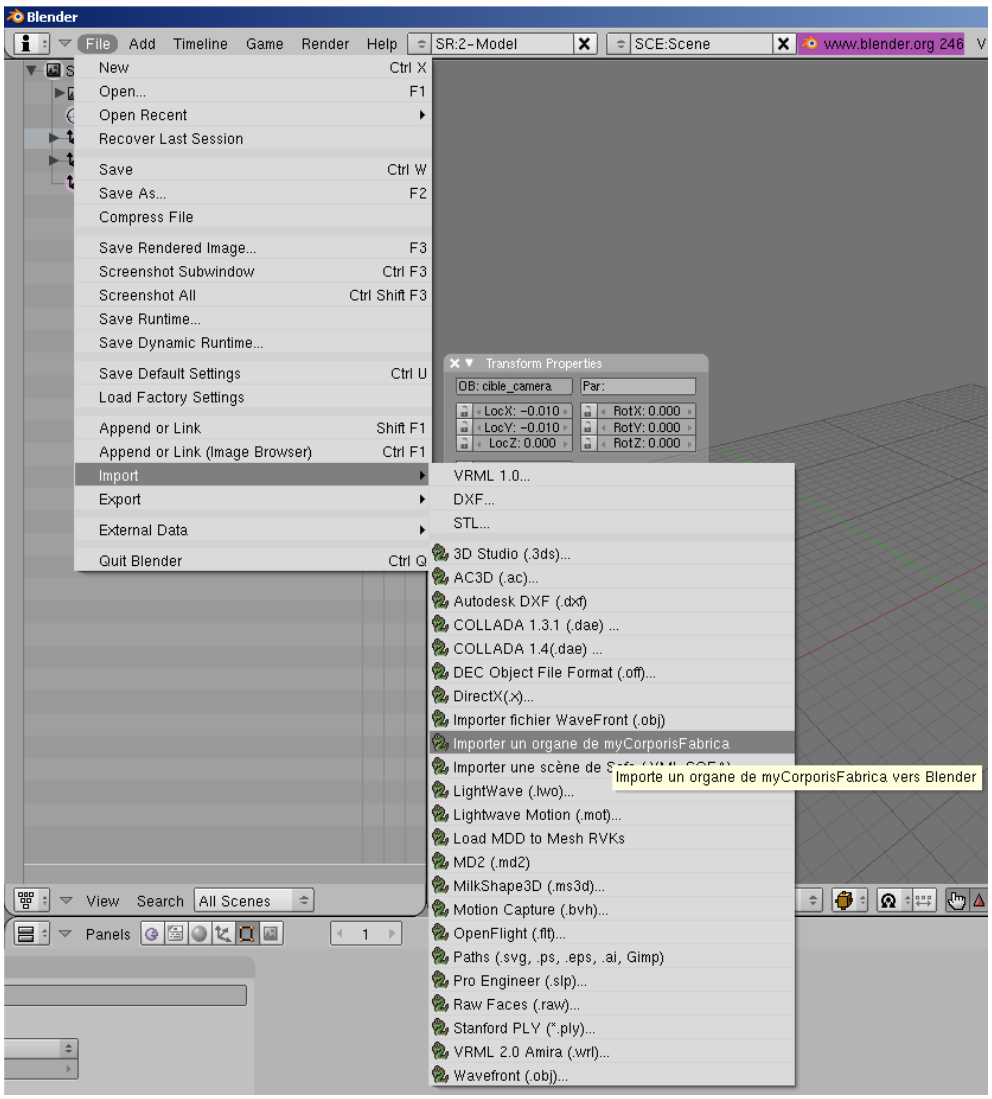


FIG. 2.41 – Appel des imports de la base à partir du menu "File - Import" de Blender

FIG. 2.42 – Écran d'accueil de Blender pour l'import des organes de la base myCorporisFabrica

FIG. 2.43 – Choix d'un organe parmi ceux disponibles (la liste est mise à jour à partir de la base myCorporisFabrica)

FIG. 2.44 – Choix d'une acquisition pour l'organe sélectionné (parmi celles disponibles dans la base de données, pour l'organe sélectionné)

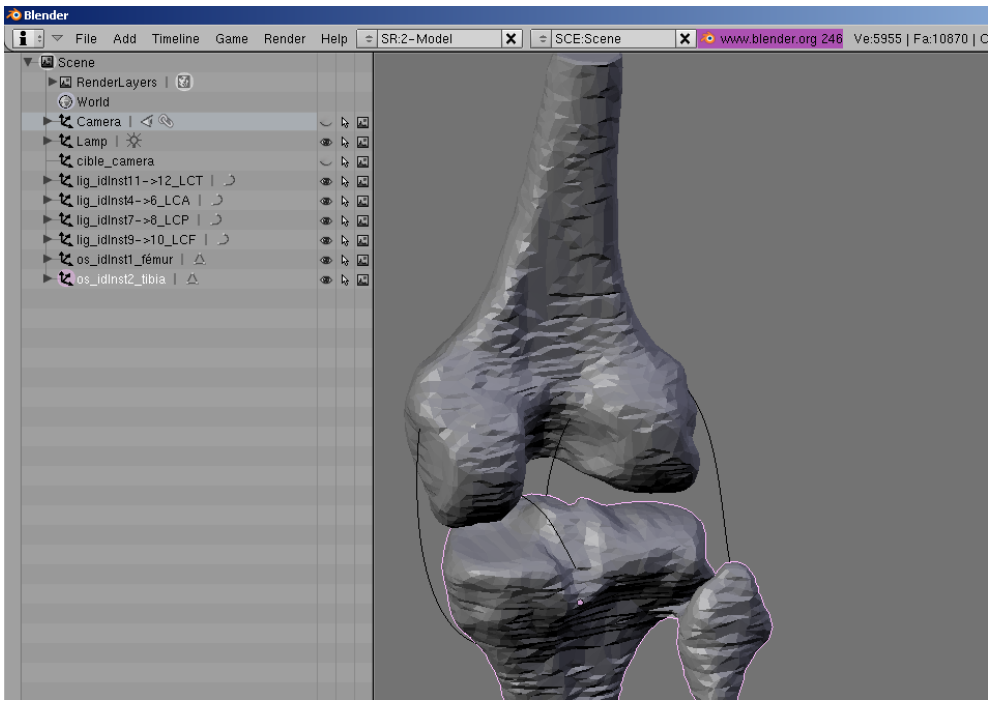


FIG. 2.45 – Résultat de l'import (tous les éléments sont déjà en place)

2.1.11 Export de la position des organes de Blender vers myCorporisFabrica v0.1

Une fois que les éléments de la scène sont placés correctement, on peut mettre à jour la base de données avec ces nouvelles positions.

L'export s'effectue simplement en utilisant le menu "File - exporter vers la base myCorporisFabrica", sans avoir besoin de préciser l'organe cible (voir figure 2.46).

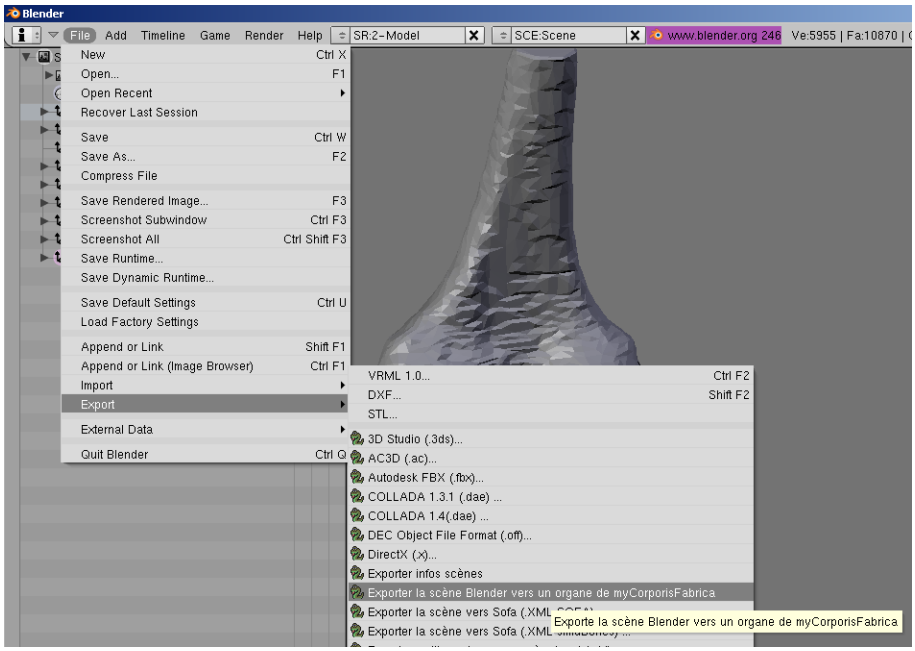


FIG. 2.46 – L'organe sélectionné importé dans Blender

Le dialogue suivant apparaît dans Blender :

FIG. 2.47 – Choix de l'export ou non de la position des ligaments

Les coordonnées des extrémités des ligaments sont configurées dans la base dans le repère local de l'élément auquel il sont attachés.

Si on décoche l'option "Exporter la position des ligaments", cela permet de déplacer un os dans Blender sans avoir à faire suivre tous les ligaments qui lui sont attachés. Lors de l'import suivant, tous les ligaments retrouvent leur point d'attache original sur l'os.

FIG. 2.48 – Message en fin de mise à jour de la base de données

2.1.12 Export de myCorporisFabrica v0.1 vers une scène SOFA

Pour écrire le programme de construction des scènes SOFA à partir des données de la base myCorporisFabrica v0.1, je me suis encore servi du langage Python, couplé à Qt (http://trolltech.com/products/qt) pour l'interface graphique. J'ai choisi de nouveau Python car l'accès à la base mySQL est facile à réaliser et cela me permettait de réutiliser le code Python déjà développé pour Blender. SOFA se sert déjà de Qt pour l'interface graphique, j'ai donc récupéré la même librairie pour ce programme. L'installation de Qt pour Python est détaillée en annexe, section 4.11, page 88.

Ce programme d'export permet de générer une scène xml SOFA en seulement trois étapes, décrites avec les figures 2.49 à 2.51 :

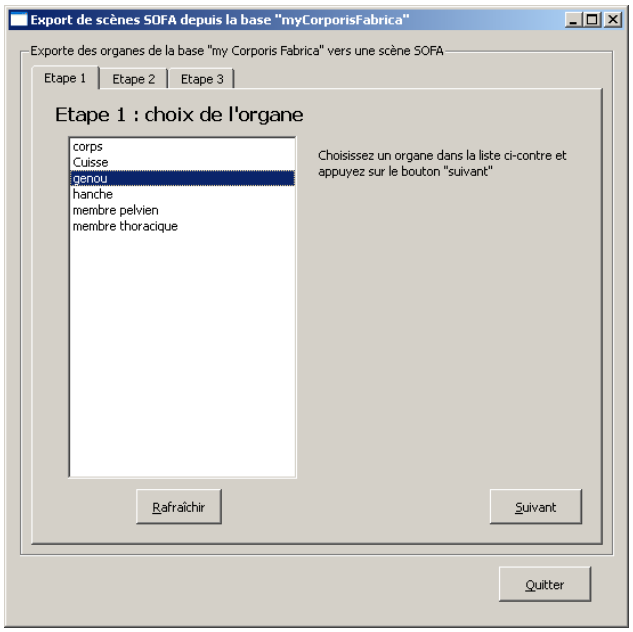


FIG. 2.49 – Choix de l'organe à exporter parmi ceux présents dans la base de données

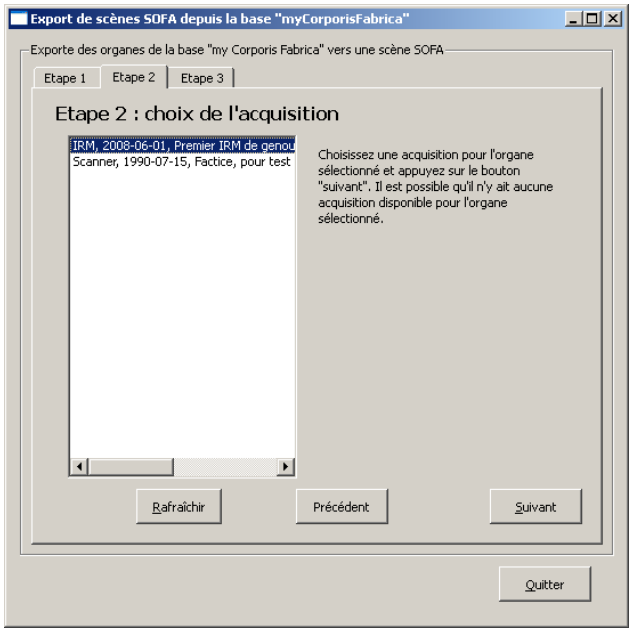


FIG. 2.50 – Choix d'un type d'acquisition pour l'organe sélectionné (scanner, IRM, etc.)

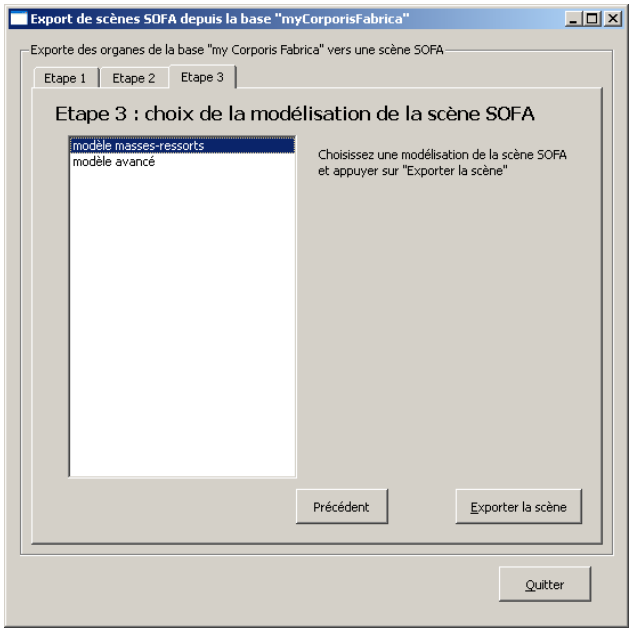


FIG. 2.51 – Choix de la modélisation de la scène pour SOFA (choix masse-ressorts)

Après l'appui sur le bouton "Exporter la scène", figure 2.51, le programme d'export génère un fichier scène XML-SOFA et ouvre SOFA (voir figures 2.52 et 2.53 page 38.

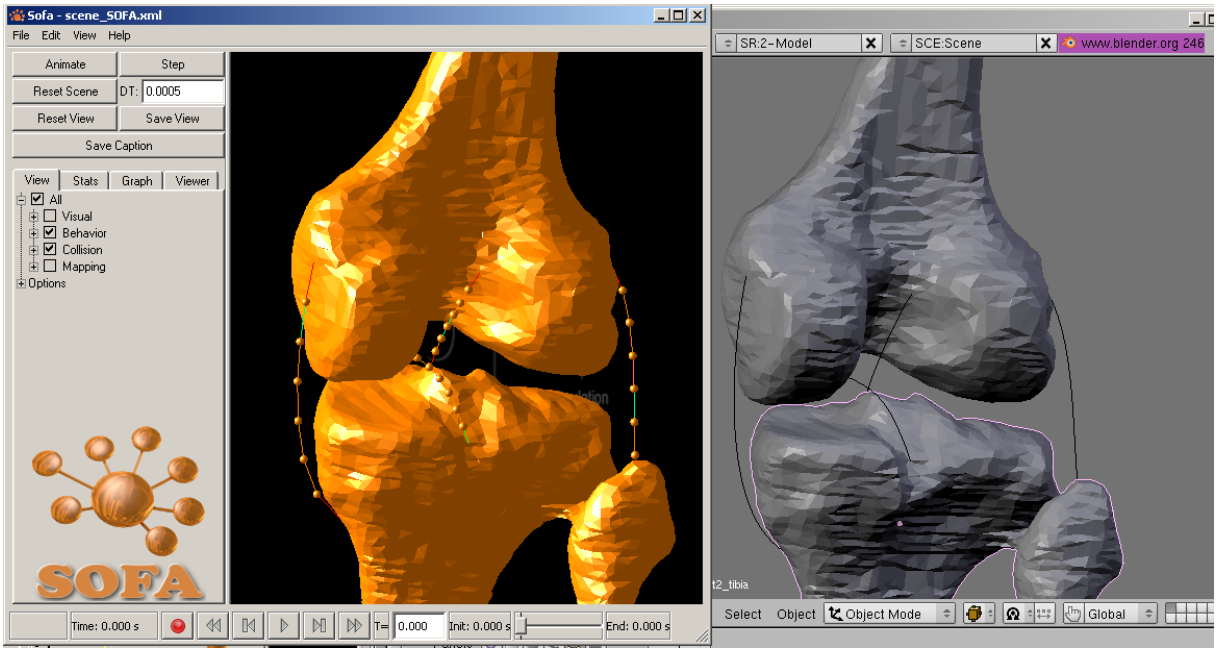


FIG. 2.52 – Vue (de dos) de la scène générée dans SOFA (à gauche) et la position des éléments réalisée avec Blender (à droite)

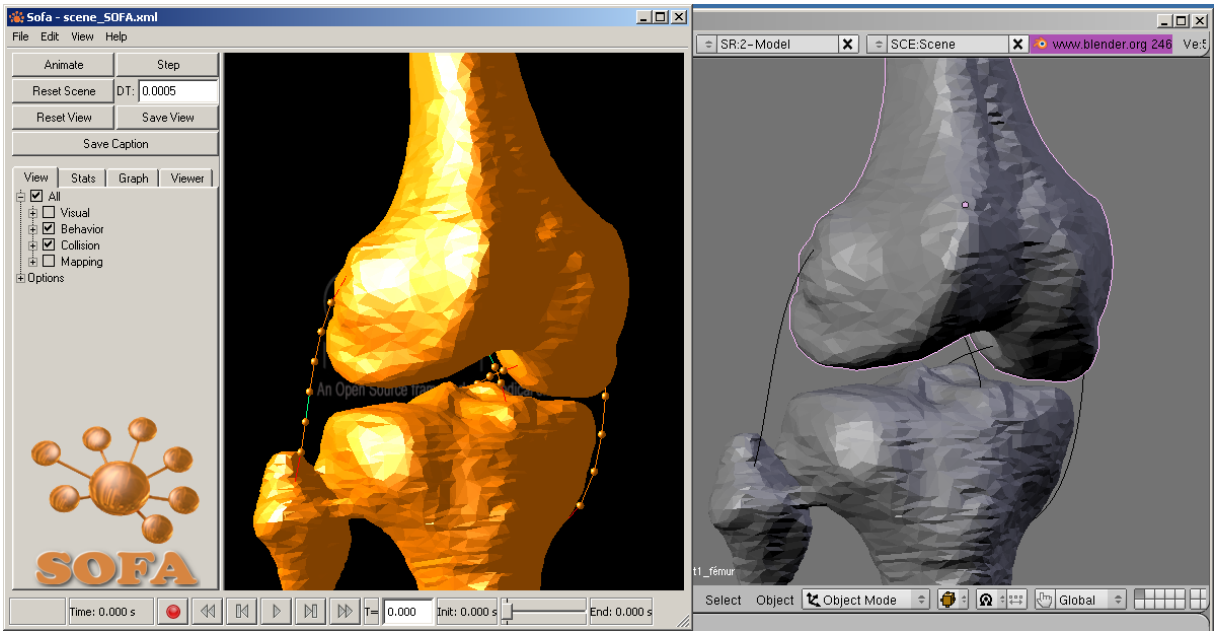


FIG. 2.53 – Vue (de face) de la scène générée dans SOFA (à gauche) et la position des éléments réalisée avec Blender (à droite)

On retrouve la même configuration que celle définie à l'origine dans Blender. La scène a été produite à partir de la base de données simplement en trois étapes et ne demande aucune connaissance particulière. On peut utiliser ce programme pour n'importe quel autre organe qui serait présent dans la base.

2.1.13 Pipe-line final de construction des scènes médicales (fin août)

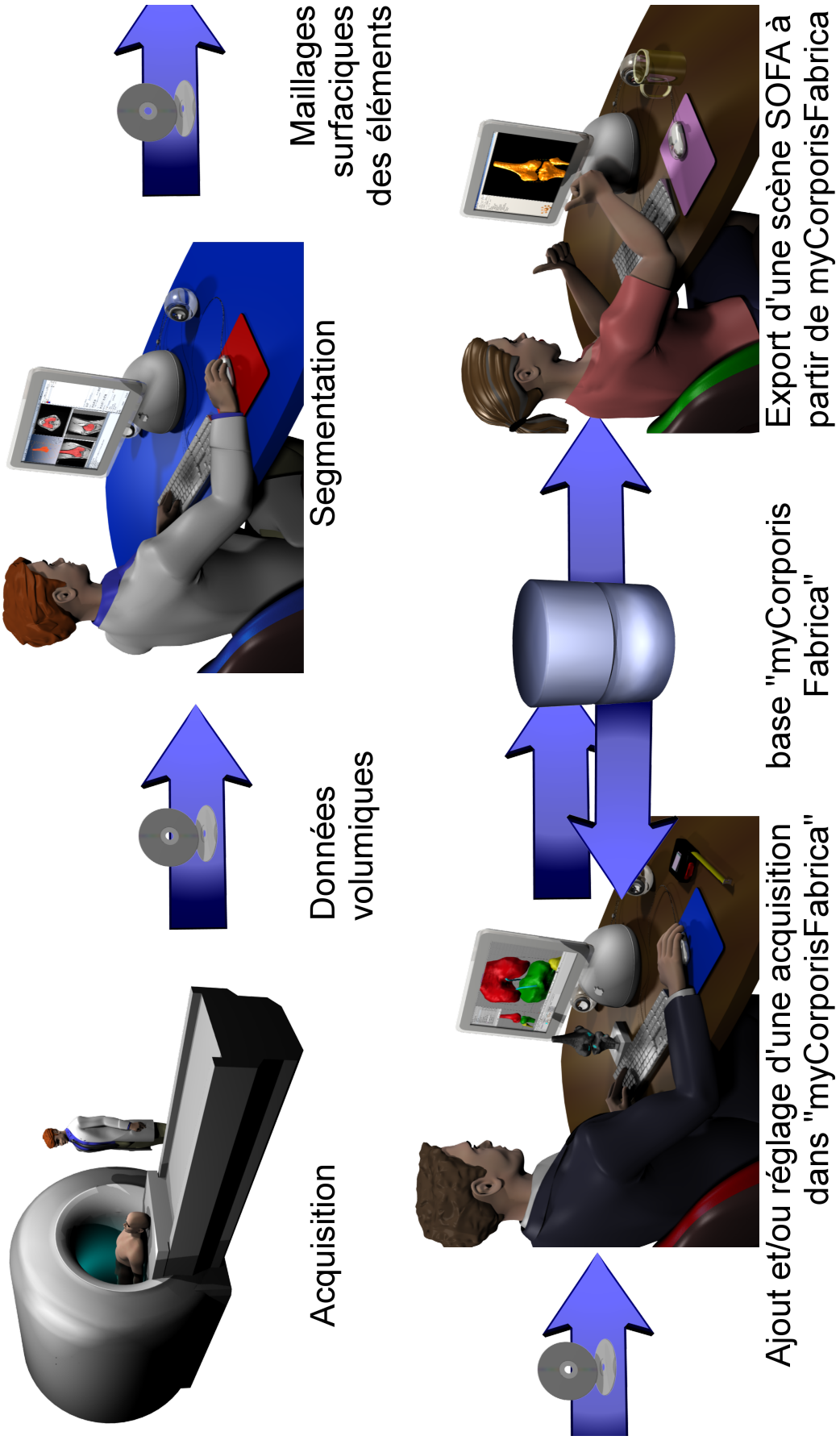


FIG. 2.54 – Pipe-line final (j'ai réalisé ces illustrations à partir des éléments 3D du site www.katorlegaz.com, sous licence Creative Commons : http://creativecommons.org/licenses/by/3.0/us/)

2.2 Diagramme de Gantt (réalisé)

Lors de la finalisation du rapport (avant la fin du stage), le diagramme de Gantt du travail réalisé était le suivant :

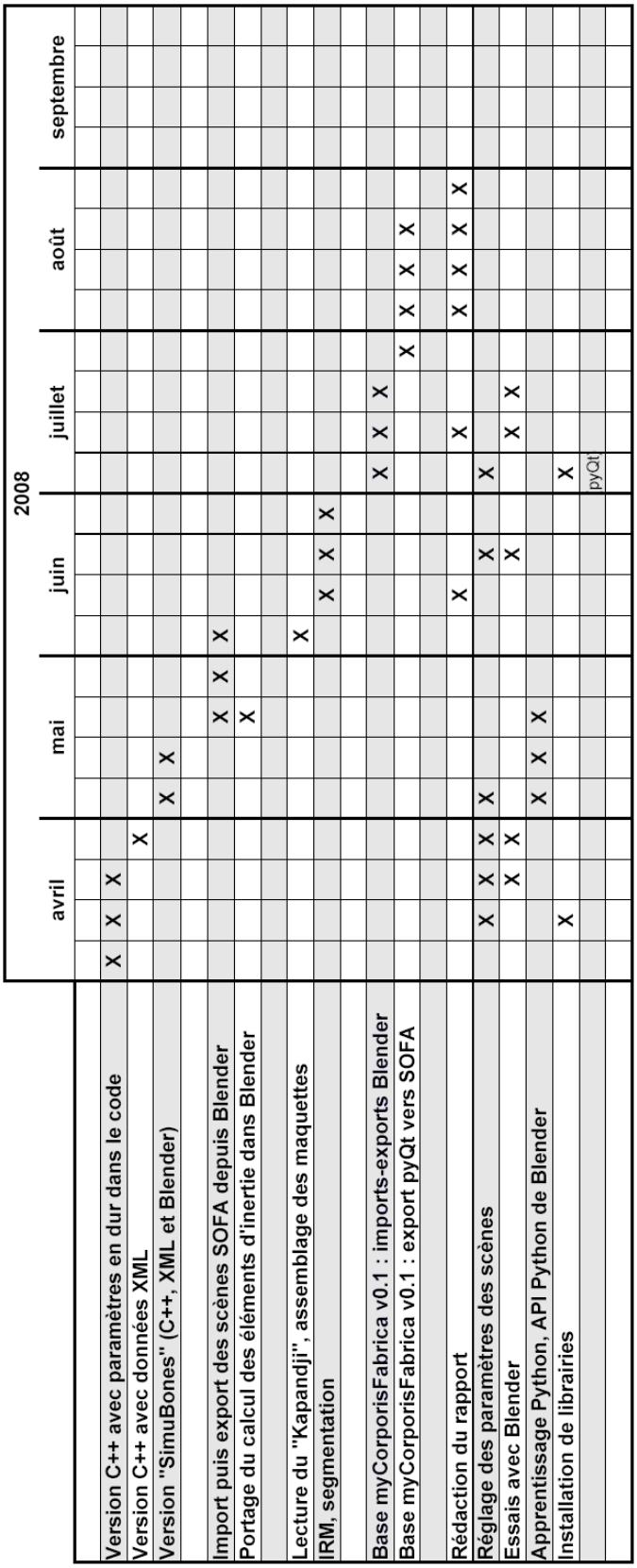


FIG. 2.55 – Diagramme de Gantt

Chapitre 3

Conclusion

3.1 Conclusion technique

Ce stage a permis d'aboutir à un pipeline (voir figure 2.54, page 39) de génération de scènes médicales. J'ai mis en place des outils qui permettent de configurer les positions des éléments dans les scènes SOFA, de configurer les positions et les inerties des éléments de la base de données "myCorporisFabrica" (créée par Olivier Palombi) et d'exporter des scènes SOFA directement à partir de cette base de données.

L'architecture finale mise en place avec Olivier Palombi et François Faure (base "myCorporisFabrica", extensions pour Blender et programme d'export) permettra donc de modéliser bien d'autres organes que le genou. Ce pipeline de construction des scènes qui n'utilise que des logiciels libres est très efficace. En effet, un utilisateur qui dispose des maillages volumiques des différents éléments d'un organe (acquis lors d'un scanner, IRM, échographie, puis d'une segmentation) pourra les ajouter dans la base "myCorporisFabrica", les placer visuellement, grâce à Blender, et exporter une scène SOFA sans écrire une seule ligne de code ni avoir à modifier aucun fichier de configuration manuellement.

Ce résultat bénéficiera à Évasion, dont un des objectifs est de travailler sur d'autres organes. Par contre, le temps passé sur la partie "construction de la scène et mise en place des éléments", a laissé peu de temps au travail de modélisation avec SOFA, ce qui était un peu plus attendu par François Boux de Casson.

Pour obtenir des scènes médicales réalistes il reste encore du travail au niveau de la modélisation physique avec SOFA. Il faudrait améliorer les réglages des ligaments (déterminer leur masse, raideur, améliorer leur comportement en compression), puis effectuer des tests de tiroir, de torsion, d'efforts et essayer éventuellement d'autres modélisations que les masses-ressorts pour les ligaments.

D'autre part, malgré mes efforts pour la segmentation, le modèle géométrique actuel des os n'est pas optimal. Un étudiant d'Olivier Palombi devrait obtenir un résultat meilleur que celui que j'ai obtenu et le réalisme de la modélisation du genou devrait en être amélioré.

Pour l'instant, les maillages sont des fichiers dont le chemin est spécifié par la base de données. Il faudra trouver un moyen de les rendre accessibles par internet.

La voie est ouverte pour poursuivre par la suite la modélisation de la partie active du genou, avec le tonus musculaire. Le placement des ligaments sur les os pourra aussi être automatisé à partir du placement des ligaments sur un maillage type.

Il reste potentiellement encore beaucoup de travail sur ce sujet, mais la direction prise lors de ce stage paraît excellente.

3.2 Conclusion générale

Ce séjour chez Évasion était très intéressant, il m'a fait découvrir la complexité de la modélisation du monde du vivant avec l'exemple du genou et la subtilité de la "conception" du corps humain. Une des phrases de conclusion du livre de A.I. Kapandji [1] sur le chapitre du genou était : "Au terme de ce chapitre, il apparaît que la stabilité du genou, articulation faiblement emboîtée, tient du miracle permanent".

J'ai aussi appris à utiliser un outil de segmentation (Amira) et un outil de dessin 3D (Blender). Blender m'a poussé à apprendre Python, son langage de programmation. J'ai été agréablement surpris de pouvoir l'utiliser pour programmer de nombreuses fonctionnalités en relation avec d'autres librairies, tout en utilisant les possibilités de l'API Blender :
– NumPy, pour les structures matrices, vecteurs ;
– Scipy, qui s'appuie sur Numpy pour l'algèbre linéaire ;
– l'accès à des fichiers XML en lecture-écriture ;
– mySQLDB, pour l'accès à une base de données mySQL ;
– pyQt pour utiliser des interfaces graphiques Qt avec Python (mais sans Blender).

Avec blender, j'ai réalisé un travail qui servira en dehors du périmètre de ce stage (tutoriels, portage du calcul des éléments d'inertie, modification des scènes SOFA).

Mon passage dans les locaux de l'Inria m'a permis d'assister à plusieurs présentations. J'ai été impressionné par celles de Yoshinori Dobashi (université d'Hokkaido), de Jerri Tessendorf (studios Rhythm & Hues), d'Antoine Bouthors (thèse sur le rendu de nuages en temps réel).

Enfin, j'ai découvert un peu le monde de la recherche à l'Inria MontBonnot (et ses brillants chercheurs) et un peu Aesculap par mes visites mensuelles chez eux.

Chapitre 4

Annexes

4.1 Organisation du projet

Cette partie indique à mon (mes) successeur(s) où se trouve le travail que j'ai réalisé et les données que j'ai utilisé.

4.1.1 Organisation générale

Je travaille sur la partition C : du disque dur de la machine "Turbulence" du bureau H101. Le système d'exploitation est Windows XP SP2. Je sauvegarde quotidiennement les deux dossiers figure 4.1 sur le réseau, à cet emplacement : "\\ evasion-home \ vansuyt \ _Sauvegarde".

Toutes mes données sont réunies dans seulement deux répertoires à la racine de C :\\ (voir figure 4.1).

– La branche `_softs` contient les logiciels libres que j'ai téléchargé, la documentation et les tutoriels associés
– La branche `_Stage_VV` contient le travail que j'ai réalisé, ainsi que la documentation et les données récoltées pour travailler sur le sujet.

```
_A --------------
_Softs
_Stage_VV
_Z --------------
```

FIG. 4.1 – Racine des données

Les sous-sections suivantes entrent plus en détail dans cette arborescence.

4.1.2 Données de la base de données "myCorporisFabrica"

C'est une base de données mySQL. L'installation de mySQL sur mon poste est faite simplement en installant EasyPhp.

Pour installer la base myCorporisFabrica, il faut utiliser un des fichiers .sql présent dans le répertoire figure 4.2.

```
_Stage_VV
   + Code
      + Bdd
         + v0.1
            + mycorporisfabrica_2008-07-04.sql
            + mycorporisfabrica_2008-07-10.sql
            + mycorporisfabrica_2008-07-25.sql
            + mycorporisfabrica_2008-07-29.sql
```

FIG. 4.2 – Fichiers sql à importer pour créer la base

L'import des données dans la base mySQL s'effectue grâce au menu easyPhp - administration (voir figure 4.3) et ensuite en appelant la fonction d'import d'un fichier .sql dans une des bases de données (voir figure 4.4). La première version de la base (v0.1) ne contient pas encore la spécification des régions où attacher les insertions des différents éléments.

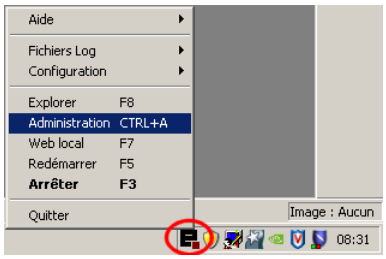


FIG. 4.3 – Appel du menu d'administration easyPhp

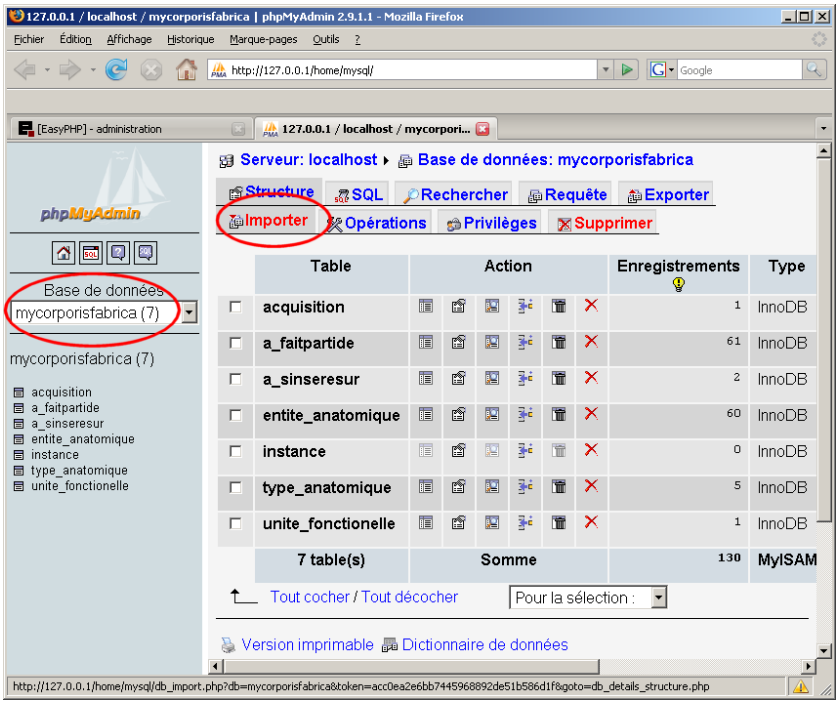


FIG. 4.4 – Import des fichier sql dans la base mySQL

4.1.3 Données 3D

Tous les maillages sont dans l'arborescence ci-dessous (voir figure 4.5).

Les dossiers "Imports dans Blender" représentent l'import des maillages originaux dans Blender.

Les dossiers "Conversion au format OBJ" représentent les maillages utilisables dans les scènes SOFA, avec les normales recalculées correctement et le centre de gravité positionné avec l'algorithme de Brian Mirtich.

```
_Stage_VV
   + Maillages
      + Livraison premiers os François Boux de Casson
         + Imports dans Blender
         + Conversion au format format OBJ
      + Données brutes CHU - IRM Genou Anthonin
         + CD_1s2
         + CD_2s2
      + Resultats_segmentation_Amira
         + Jeu 00 CD 2 - Vincent
            + Imports dans Blender
            + Conversion au format format OBJ
      + Rotules de test
```

FIG. 4.5 – Projets C++

Dans le dossier `Jeu 00 CD 2 - Vincent \ Imports dans Blender`, on trouve des fichiers version 1 et version 2 :
– La version 1 correspond à une première segmentation avec un tibia sans fibula.
– La version 2 correspond à une reprise de la segmentation version 1 (avec le même jeu de données), mais avec la fibula attachée au tibia et avec des cartilages un peu plus épais sur les glènes (surfaces de contact sur le plateau tibial).

4.1.4 Scènes

Les scènes se découpent en deux catégories : celles pour simubones et celles pour SOFA (voir figure 4.6).

Les scènes pour mon code “SimuBones” répondent au schéma XSD suivant :
`C:\_Stage_VV\Sofa\branches\Sofa-1.0\scenes_VV\scenes_simuBones_v0_03\scene.xsd`

Le dossier `_Stage_VV\Scenes Scenes_Simubones_Blender` ne contient que les fichiers “Blender” des scènes simuBones (voir la section “Projets C++ réalisés” (page 46)).

Les dossiers `_Stage_VV\Scenes\Scenes_SOFA` contiennent les fichiers XML des scènes SOFA (sans programme intermédiaire), plus les fichiers Blender correspondants.

```
_Stage_VV
   + Scenes
      + Scenes_Simubones_Blender
         + Scene_rotules
         + Scènes_maillage_IRM
      + Scenes_SOFA
         + Casse-noix
         + ChainAll
         + Genou_01
         + Genou_02
         + Genou_03
```

FIG. 4.6 – Scènes

4.1.5 Développements

Projets C++ réalisés

Voici, figure 4.7, la liste des projets C++ réalisés pour SOFA, par ordre chronologique.

Les projets C++ ont été réalisés en début de stage et ont étés abandonnés par la suite en fonction de l'évolution de l'architecture du logiciel.

Les projets "rotules_v1_2" à "rotules_v1_6" sont les premiers essais réalisés avec deux maillages composés par l'union d'un cylindre et d'une sphère en guise de tibia et fémur.

Ensuite, le projet "genou_v0_01" est une dérivation de "rotules_v1_6" avec la première version des maillages tibia-fémur.

Enfin les projets "simuBones_v0_01" à "simuBones_v0_03" sont les programmes qui fonctionnent avec une scène externe au format XML avec mon propre format de fichier scène.

```
_Stage_VV
   + Sofa
      + branches
         + Sofa-1.0
            + applications
               + tutorials
                  + rotules_v1_2
                  + rotules_v1_3
                  + rotules_v1_4
                  + rotules_v1_5
                  + rotules_v1_6
                  + genou_v0_01
                  + simuBones_v0_01
                  + simuBones_v0_02
                  + simuBones_v0_03
            + bin
            + scenes
            + scenes_VV

         + Sofa-dev
```

FIG. 4.7 – Projets C++

Les arborescences "Sofa-1.0" et "Sofa-dev" contiennent toutes les deux les arborescences "Sofa", mais l'arborescence "Sofa-1.0" a été figée pour servir de référence par rapport à la dernière version ("Sofa-dev") et pour être sûr que mes projets de début de stage fonctionnent toujours.

L'arborescence "Sofa-dev" est synchronisée régulièrement avec le développement en cours de Sofa et ne contient aucune scène ni aucun développement.

Code Python pour Blender

```
_Stage_VV
   + Code
      + scripts_python_Blender
```

FIG. 4.8 – Projets Python pour Blender

- ajout de méthodes de lecture-écriture de fichiers 3D obj au projet réalisé en cours d'année (Projet proposé par Cédric Gérot et Jean-Claude Léon en Master 2 Mathématiques et Informatique, option IICAO)
- calcul d'une surface offset
- conversion de fichiers au format obj en fichier 3D off (pour pouvoir utiliser un utilitaire qui affiche les numéros des sommets et de faces, pour la mise au point de programmes)
- conversion des fichier du format `obj` en format `polyhedron` pour pouvoir les utiliser dans le programme de Bryan Mirtich (calcul du centre de gravité et des éléments d'inertie)
- écriture d'un programme de projection de sommets sur un maillage (pour pouvoir placer les points d'attache plus facilement sur les os)
- création d'un programme "SimuBones" de construction d'une scène SOFA à partir d'un fichier XML.
- écriture de petits scripts Python pour Blender pour faciliter le traitement des maillages
  - export dans un fichier texte de la liste des faces sélectionnées sur un maillage
  - renvoi des informations sur une sélection dans un maillage (numéros de faces, de sommets)
  - renvoie de la distance entre deux sommets
  - projection des sommets sélectionnés sur un plan de côte z = valeur
- ajout de scripts d'import-export de scènes XML SOFA dans Blender
- portage du code de Brian Mirtich ("VolInt.C") en Python pour Blender (ce code calcule la position du centre de gravité, la matrice d'inertie 3x3 par rapport au centre de gravité)
- import des fichiers VRML 2.0 exportés par Amira dans Blender (dérivé de l'import existant des fichiers OFF)

Code Python hors Blender

```
_Stage_VV
   + Code
      + workspace_Eclipse
         + Exports_scenes_de_myCorporisFabrica
         + Parcours_XML
         + Test_Bezier
```

FIG. 4.9 – Projets Python hors Blender

Code C++ console pour le traitement des maillages

Les programmes C++ figure 4.10 ont aussi été écrits en début de stage, principalement pour réaliser des petits traitement sur les maillages alors que je connaissais assez mal Blender.

```
_Stage_VV
   + Code
      + preparation maillages
         + CnvObjToOff
         + CoupeObj
         + LibMaillage
         + ObjToPolyhedron
         + ProgSubdivision_TRIANGLES
         + Projection_points_sur_maillage
         + Surface offset
         + Translate Obj
```

FIG. 4.10 – Code C++ console

Voici leurs fonctionnalités :

- `CnvObjToOff` : conversion des fichiers 3D au format Obj en format Off, pour pouvoir utiliser le viewer Antiview. Existe déjà avec Blender et ma lecture-écriture de format obj sans transformations
- `CoupeObj` : conserve les sommets et faces contenues dans un paralèllépipède défini par deux points
- `LibMaillage` : c'est une librairie sur les maillages issue du projet C++ sur la subdivision des maillages écrite en cours d'année M2-IICAO
- `ObjToPolyhedron` : un programme de conversion des fichiers 3D Obj vers le format Polyhedron pour tester le programme de Brian Mirtich sur le calcul du centre de gravité des maillages ainsi que leurs éléments d'inertie
- `ProgSubdivision_TRIANGLES` : c'est le programme réalisé en "projets Master 2". Il permet de créer plusieurs subdivisions (par Loop) d'une sélection de faces d'un maillage triangulaire. Je m'en suis servi pour lisser les glènes et les condyles des tibia et fémur. Blender ne fait pas encore ce genre de subdivisions (seulement Catmull-Clark, à ma connaissance). La sélection des numéros des faces est facilement faite avec Blender et exportée dans un fichier grâce à un script Python écrit pendant le stage.
- `Projection_points_sur_maillage` : ce programme parcours les points d'attache d'une scène XML "simuBones" et les projette sur le maillage correspondant. J'avais écrit ce programme quand je n'utilisais pas encore Blender pour placer les points d'attache des ligaments.
- `Surface offset` : calcule la surface offset d'un maillage. Il prend en argument un maillage et une distance. Je m'en suis servi pour décaler la surface d'un maillage pour tester la présence ou non d'un cartillage et générer des ménisques au début du stage. Le résultat de l'expérience n'était pas concluant.
- `Translate Obj` : ce programme m'a juste servi à translater tous les sommets d'un maillage (pour que le zéro soit au centre de gravité). On peut facilement le faire avec Blender.

Tous ces petits programmes affichent leur mode d'emploi juste en les appelant sans arguments.

4.2 Détail du premier modèle d'articulation avec SOFA

C'est le modèle appelé "Rotules v1.3", codé entièrement en C++, avec les paramètres de la scène en dur dans le code. Il est composé de deux solides (en rouge et bleu sur les figures ci-dessous). Le solide rouge (celui de gauche) est fixe, le solide bleu (à droite) est soumis à la gravité un ressort entre les "petits" repères (voir figures 4.13 et 4.11) a été ajouté.

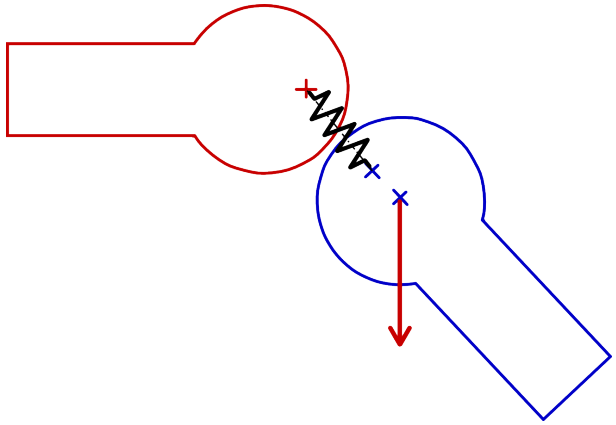


FIG. 4.11 – Schéma

4.2.1 Résultat obtenu avec SOFA

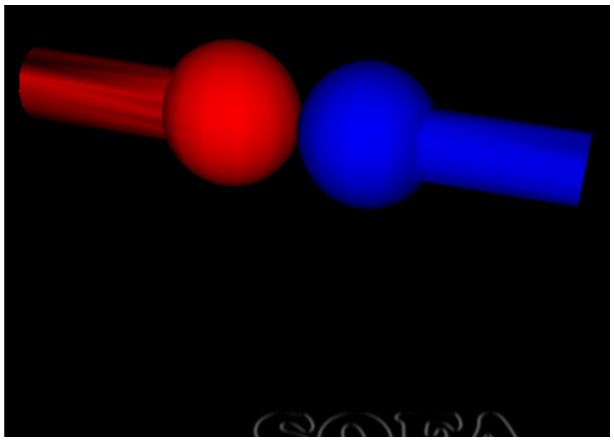


FIG. 4.12 – Solides en position initiale

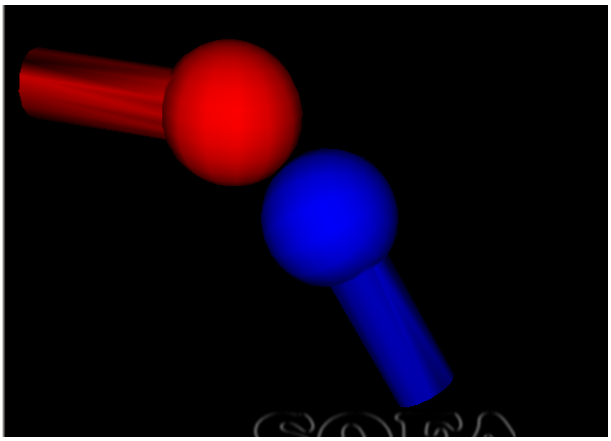


FIG. 4.13 – Solides en position finale

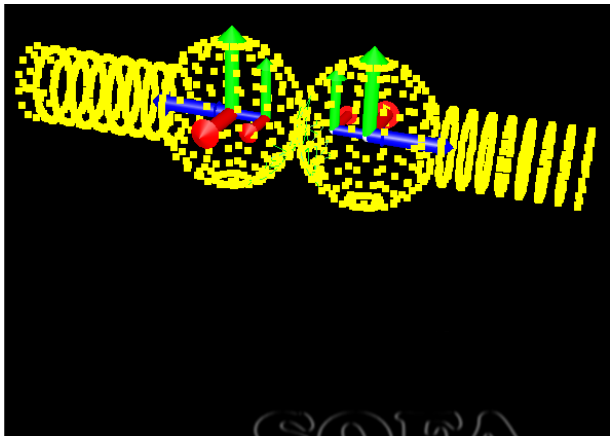


FIG. 4.14 – Solides en position initiale (vue mécanique)

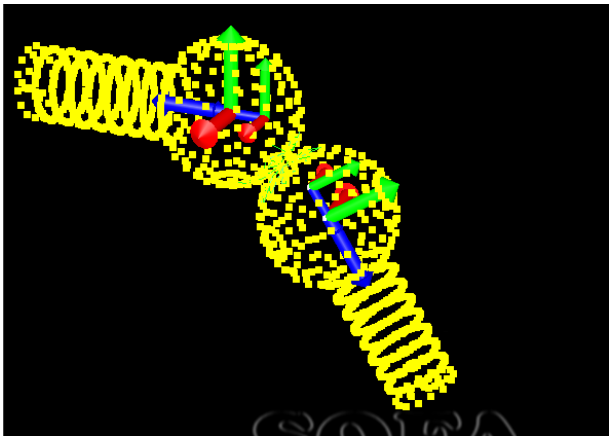


FIG. 4.15 – Solides en position finale (vue mécanique)

4.2.2 Graphe de scène correspondant

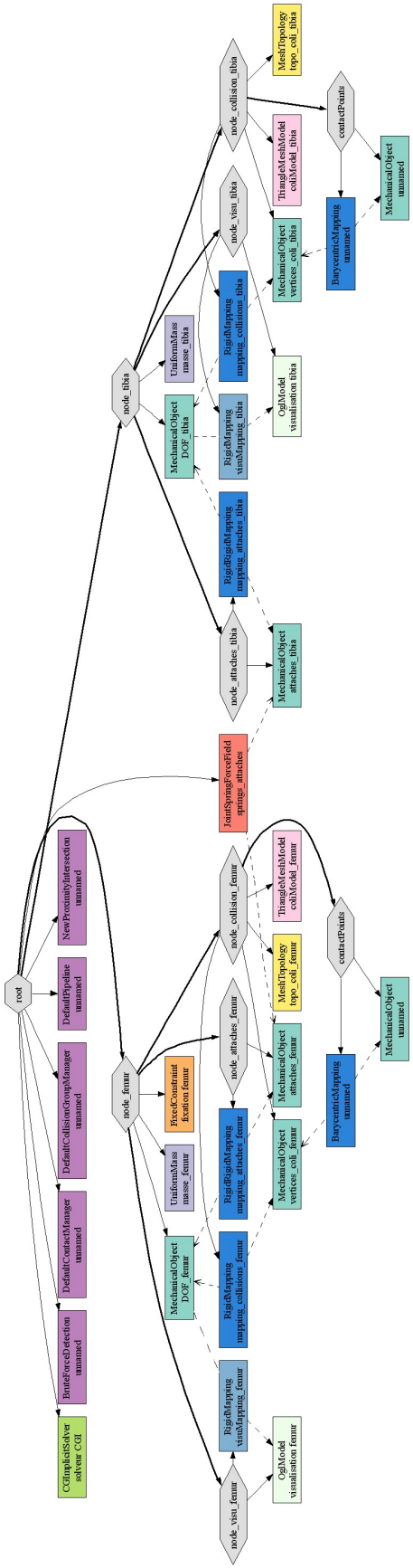


FIG. 4.16 – Graphe de scène

4.3 Détail du logiciel “SimuBones”

Ce programme de transition n’est pas retenu pour la suite du projet, je ne passe donc pas beaucoup de temps pour sa description.

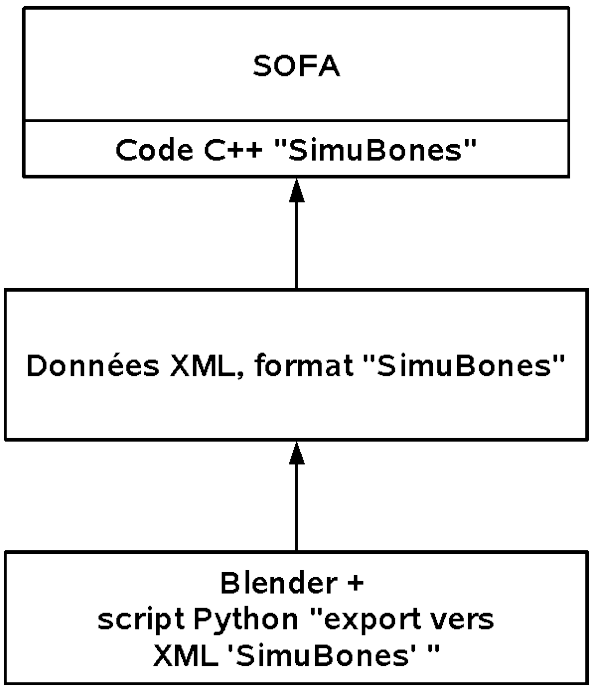


FIG. 4.17 – Schéma de fonctionnement

4.3.1 Détail du fichier XML de configuration

Le squelette du fichier XML correspond à ceci :

```
<scene xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" nom="genou">
  <!-- Configuration de SOFA -->
  <configuration_SOFA>
    <NewProximityIntersection alarm_distance="0.6" contact_distance="0.20" />
  </configuration_SOFA>
  <!-- Informations pour les os -->
  <elements_rigides>
    <os nom="femur" fixe="1" ...>
      ...
      <attaches>
        <attache nom="LLI_sur_femur" x="-6.0" y="-19.5" z="-0.6" vx="-0.3" vy="
            -1.0" vz="0" />
        <attache nom="LLE_sur_femur" x="2" y="-17" z="-0.99" vx="2.0" vy="-1" vz="
            0" />
        <attache nom="LCA_sur_femur" x="-0.2" y="-20.7" z="-1.2" vx="-0.6" vy="0.3
            " vz="0.4" />
        <attache nom="LCP_sur_femur" x="-2.1" y="-21.4" z="1.0" vx="0.2" vy="-0.4"
             vz="-0.7" />
      </attaches>
    </os>
    <os nom="tibia" fixe="0" masse="2.0">
      ...
      <attaches>
        <attache nom="xxx1" x="x" y="x" z="x" vx="x" vy="x" vz="x" />
        <attache nom="xxx2" x="x" y="x" z="x" vx="x" vy="x" vz="x" />
        <attache nom="xxx3" x="x" y="x" z="x" vx="x" vy="x" vz="x" />
```

```
      </attaches>
    </os>
  </elements_rigides>
  <elements_souples>
    <ligament nom="x1" masse="x" ...>
      <attache_source nom="xxx1" />
      <attache_destination nom="yyy1" />
    </ligament>
    <ligament nom="x2" ... </ligament>
    <ligament nom="x3" ... </ligament>
  </elements_souples>
</scene>
```

C’est une définition d’éléments rigides (les os), sur lesquels on place une série d’attaches nommées, dans le repère de l’os. Sur chaque attache est assigné un vecteur directeur qui indique un direction de départ de l’élément fixé sur cette attache.

Ensuite, dans “éléments souples”, on définit une liste de ligaments avec leurs propriétés et les noms des attaches (uniques dans la scène) sur lesquels ils sont attachés.

Le détail complet d’un des premiers fichiers complet avec juste un tibia, un fémur et les ligaments LCA, LCP, LLI et LLE donne le listing suivant 4.1.

Listing 4.1 – Listing SimuBones

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
SCENE GENOU avec os Francois Boux de Casson
Version v0_3
 -->
<scene xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" nom="genou">
  <!-- Configuration de SOFA -->
  <configuration_SOFA>
    <NewProximityIntersection alarm_distance="0.6" contact_distance="0.20" />
  </configuration_SOFA>
  <!-- Informations pour les os -->
  <elements_rigides>
    <!-- Informations pour le femur -->
    <os nom="femur" fixe="1" masse="1.0">
      <couleur r="0.7" g="0.7" b="0.7" a="1.0" />
      <contact raideur="20" friction="1" />
      <fichier_maillage_peau nom="VisualModels/femur_L_cdg_cut_ref_export.obj" />
      <fichier_maillage_collision nom="CollisionModels/femur_L_cdg_collision.obj"
          />
      <position_lineaire x="0.0" y="0.0" z="0.0" />
      <position_angulaire rx="-90" ry="0" rz="0" />
      <matrice_inertie A="1" B="1" C="1" D="0" E="0" F="0" />
      <attaches>
        <attache nom="LLI_sur_femur" x="-6.0" y="-19.5" z="-0.6" vx="-0.3" vy="
            -1.0" vz="0" />
        <attache nom="LLE_sur_femur" x="2" y="-17" z="-0.99" vx="2.0" vy="-1" vz="
            0" />
        <attache nom="LCA_sur_femur" x="-0.2" y="-20.7" z="-1.2" vx="-0.6" vy="0.3
            " vz="0.4" />
        <attache nom="LCP_sur_femur" x="-2.1" y="-21.4" z="1.0" vx="0.2" vy="-0.4"
             vz="-0.7" />
      </attaches>
    </os>
    <!-- Informations pour le tibia -->
    <os nom="tibia" fixe="0" masse="2.0">
      <couleur r="0.8" g="0.8" b="0.8" a="1.0" />
      <contact raideur="20" friction="1" />
      <fichier_maillage_peau nom="VisualModels/tibia_L_cdg_cut.obj" />
```

```
34        <fichier_maillage_collision nom="CollisionModels/tibia_L_cdg_collision.obj"/
              >
          <position_lineaire x="-0.5" y="-1" z="38.5"/>
36        <position_angulaire rx="-90" ry="2" rz="0"/>
          <matrice_inertie A="1" B="1" C="1" D="0" E="0" F="0"/>
38        <attaches>
            <attache nom="LLI_sur_tibia" x="-4.5" y="15" z="-0.0" vx="-0.2" vy="1.0"
                vz="0"/>
40          <attache nom="LLE_sur_tibia" x="4.0" y="15" z="-0.5" vx="1" vy="1" vz="0"/
                >
            <attache nom="LCA_sur_tibia" x="-1.5" y="16.5" z="3.0" vx="1.0" vy="0.5"
                vz="-0.4"/>
42          <attache nom="LCP_sur_tibia" x="1.0" y="17.0" z="-1.6" vx="-0.6" vy="2.4"
                vz="0.6"/>
          </attaches>
44      </os>
        <!-- Informations pour la rotule (patella) -->
46    </elements_rigides>
      <!-- Informations pour les ligaments -->
48    <elements_souples>
        <ligament nom="LLI" masse="0.1" nb_particules="14" raideur="4000" amortisseur=
            "10" rayon="0.1" longueur="3.646"><!-- longueur="3.646" -->
50        <attache_source nom="LLI_sur_femur"/>
          <attache_destination nom="LLI_sur_tibia"/>
52      </ligament>
        <ligament nom="LLE" masse="0.1" nb_particules="14" raideur="4000" amortisseur=
            "10" rayon="0.1" longueur="5.8"><!-- longueur="6.17709" -->
54        <attache_source nom="LLE_sur_femur"/>
          <attache_destination nom="LLE_sur_tibia"/>
56      </ligament>
        <ligament nom="LCA" masse="0.1" nb_particules="8" raideur="4000" amortisseur="
            10" rayon="0.1" longueur="3.26993">
58        <attache_source nom="LCA_sur_femur"/>
          <attache_destination nom="LCA_sur_tibia"/>
60      </ligament>
        <ligament nom="LCP" masse="0.1" nb_particules="12" raideur="4000" amortisseur=
            "10" rayon="0.1" longueur="4.1509">
62        <attache_source nom="LCP_sur_femur"/>
          <attache_destination nom="LCP_sur_tibia"/>
64      </ligament>
    </elements_souples>
66</scene>
```

4.3.2 Modification du fichier XML de configuration avec Blender

Devant la difficulté à modifier les position de chaque éléments géométrique du fichier 4.1, j’ai utilisé le logiciel Blender. En effet, Blender permet d’utiliser des scripts Python et de lire et écrire des fichiers XML.

Le script Blender exporte uniquement une scène vers le fichier XML de configuration. La correspondnce entre la scène Blender et le fichier XML s’effectue par le nom et le type des objets dans Blender et le nom des os et ligaments dans le fichiers XML de configuration “Simubones”. Seuls les éléments trouvés dans le fichier XML sont mis à jour, le fichier XML doit préalablement exister.

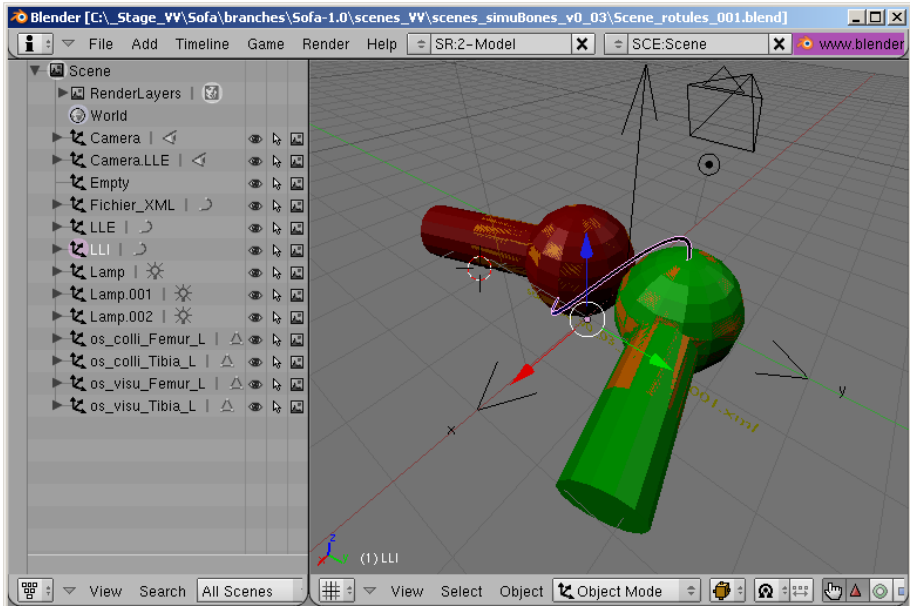


FIG. 4.18 – Configuration d’une scène SimuBones dans Blender

Le fichier XML au format “SimuBones” correspondant à cette scène est le fichier suivant :

Listing 4.2 – Listing SimuBones - rotules

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<scene nom="genou" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<!-- Configuration de SOFA -->
<configuration_SOFA dt="0.01">
  <NewProximityIntersection alarm_distance="0.5" contact_distance="0.10"/>
</configuration_SOFA>
<!-- Informations pour les os -->
<elements_rigides>
  <!-- Informations pour le femur -->
  <os fixe="1" masse="3.0" nom="Femur_L">
    <couleur a="1.0" b="0.7" g="0.7" r="0.7"/>
    <contact friction="0.0" raideur="10"/>
    <fichier_maillage_peau nom="VisualModels/os_Femur_L.obj"/>
    <fichier_maillage_collision nom="CollisionModels/os_Femur_L.obj"/>
    <position_lineaire x="0.0000" y="-1.0000" z="0.0000"/>
    <position_angulaire rx="0.0000" ry="0.0000" rz="20.0000"/>
    <matrice_inertie A="1" B="1" C="1" D="0" E="0" F="0"/>
    <attaches>
      <attache nom="LLI_sur_Femur_L" vx="1.5353" vy="0.0381" vz="0.0150" x="0.9935
          " y="-0.0091" z="-0.0369"/>
      <attache nom="LLE_sur_Femur_L" vx="-1.0577" vy="0.0460" vz="0.0644" x="
          -0.9947" y="-0.0055" z="-0.0012"/>
    </attaches>
  </os>
  <!-- Informations pour le tibia -->
  <os fixe="0" masse="3.0" nom="Tibia_L">
    <couleur a="1.0" b="0.8" g="0.8" r="0.8"/>
    <contact friction="0.0" raideur="10"/>
    <fichier_maillage_peau nom="VisualModels/os_Tibia_L.obj"/>
    <fichier_maillage_collision nom="CollisionModels/os_Tibia_L.obj"/>
    <position_lineaire x="0.0000" y="1.0000" z="0.0000"/>
    <position_angulaire rx="0.0000" ry="0.0000" rz="-75.0000"/>
    <matrice_inertie A="1" B="1" C="1" D="0" E="0" F="0"/>
    <attaches>
      <attache nom="LLI_sur_Tibia_L" vx="0.0024" vy="0.0186" vz="1.3599" x="0.0144
          " y="0.0272" z="0.9631"/>
      <attache nom="LLE_sur_Tibia_L" vx="0.0570" vy="0.0355" vz="-0.9977" x="
          0.0023" y="-0.0090" z="-1.0012"/>
```

```
    </attaches>
  </os>
</elements_rigides>
<!-- Informations pour les ligaments -->
<elements_souples>
<ligament amortisseur="1" longueur="3.13" masse="0.1" nb_particules="20" nom="LLI"
    raideur="20" rayon="0.1">
  <attache_source nom="LLI_sur_Femur_L"/>
  <attache_destination nom="LLI_sur_Tibia_L"/>
 </ligament>
 <ligament amortisseur="1" longueur="3.61" masse="0.1" nb_particules="20" nom="LLE
    " raideur="20" rayon="0.1">
  <attache_source nom="LLE_sur_Femur_L"/>
  <attache_destination nom="LLE_sur_Tibia_L"/>
</ligament>
</elements_souples>
</scene>
```

Pour illustrer le parcours de la scène Blender, voici le programme principal (“main”) du fichier-script Python d’export pour Blender (moins quelques raccourcis) :

Listing 4.3 – ”Script “main” d’export SimuBones”

```
#!BPY

"""
Name: 'Exporter la scène vers Sofa (.XML simuBones) ...'
Blender: 240
Group: 'Export'
Tooltip: 'Exporte la scène vers le fichier XML de simuBones (.XML)'
"""

__author__ = "Vincent Vansuyt"
__url__ = ("","")
__version__ = "1.0 2008/05/19"

__bpydoc__ = """\
Ce script exporte la scène vers le fichier XML "SimuBones" de Sofa.
"""

from Blender import Draw, Window, Scene, NMesh, Mathutils, sys, Object, Scene,
    Text
import Blender
import BPyMesh
[...]

#
# Exporte la scène dans le fichier XML
#
def exporter():
  print "----- debut du script -----"
  # Récupération du fichier XML à patcher et vérification de son existence
  strNomFichierXML = getNomFichierXML()
  if not os.path.isfile( strNomFichierXML ):
    strM = "Attention|Le fichier XML n'existe pas (%s)" %( strNomFichierXML )
    Draw.PupMenu( strM )
    print strM
    return
  print "Nom complet du fichier XML : %s\n"%(strNomFichierXML)

  # Parcours de la scène à la recherche des objets à traiter
  lst_obj = Blender.Object.Get()
  fxStr = Xstring()
```

```
  # - parcours des Os de la scène
41 listeNomsOs = []
  monOs = Os()
43 for obj in lst_obj:
    if obj.getType() == "Mesh":
45    strNom = obj.getName()
      if fxStr.bCommencePar( strNom, "os_" ):
47      listeNomsOs.append( strNom )
        monOs.setParametresObjet( obj )
49      monOs.setParametresXML( strNomFichierXML )
        monOs.afficher()
51      if fxStr.bCommencePar( strNom, "os_visu" ):
          monOs.mettre_fichier_XML_a_jour()
53
  # - parcours des Ligaments de la scène
55 ligament = Ligament()
  ligament.setListeNomsOs( listeNomsOs )
57 for obj in lst_obj:
    if obj.getType() == "Curve":
59    ligament.setParametresCourbe( obj )
      ligament.setParametresXML( strNomFichierXML )
61    ligament.afficher()
      ligament.mettre_fichier_XML_a_jour()
63
exporter()
```

4.4 Détail du logiciel “Import - export des scènes SOFA avec Blender”

4.4.1 Script “Import des scènes SOFA dans Blender”

Versions des logiciels utilisés

Les scripts suivant ont été écrits avec Blender 2.46 sous Windows XP pro 2002, service pack 2. Ils devraient fonctionner normalement sur toutes les machines (Linux ou Windows) et Blender à partir de la version 2.40, sans compilation.

Utilisation

Cette fonction d’import fonctionne avec un fichier XML d’initialisation (dans le dossier qui contient tous les fichiers source de la fonction d’import). Le fichier XML ci-dessous (listing 4.4) sert juste à ouvrir l’explorateur de fichiers de Blender directement dans le fichier des scènes, sans avoir à naviguer à chaque fois vers les fichiers .scn SOFA.

Listing 4.4 – Fichier XML d’initialisation “Config.xml”

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Config_Import_Sofa>
  <Chemins strChemin_scenes="C:\_Stage_VV\Sofa-dev\trunk\Sofa\scenes" />
</Config_Import_Sofa>
```

Voici la procédure à utiliser pour importer des scènes Blender dans SOFA :

1. L’accès à la fonction d’import se fait par le menu de Blender “File - Import” (voir figure 4.19)
2. Ensuite on sélectionne le fichier scène SOFA à importer (figure 4.20)
3. Après quelques secondes d’attente et rotation de la vue, on obtient le résultat 4.21)

La figure 4.22 représente le même fichier scène ouvert avec SOFA.

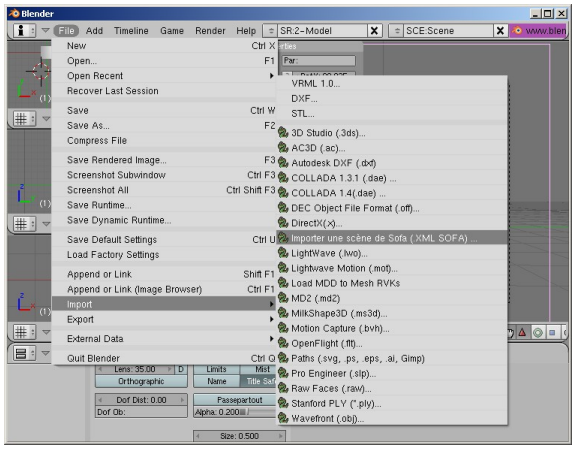


FIG. 4.19 – Accès à la fonction d’import

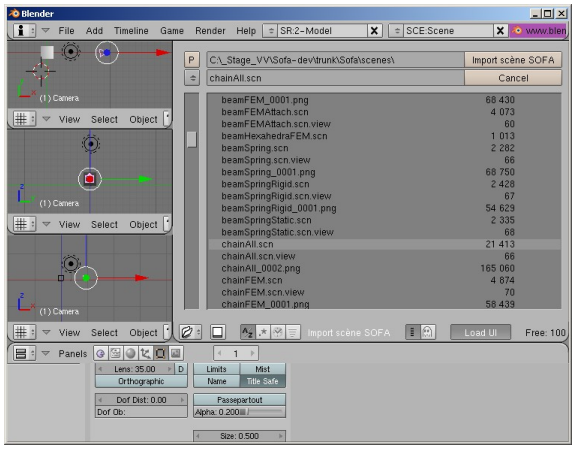


FIG. 4.20 – Choix de la scène à importer

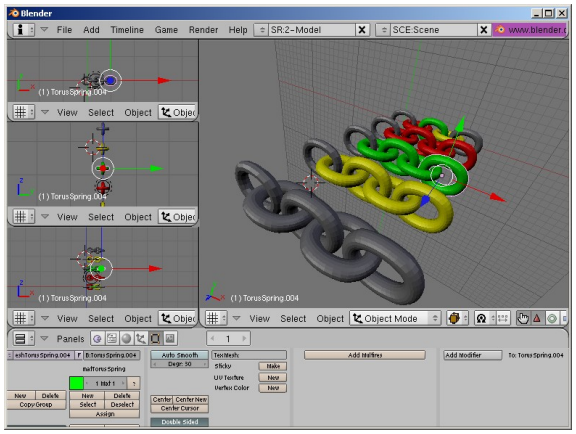


FIG. 4.21 – Résultat dans Blender (après orientation de la vue)

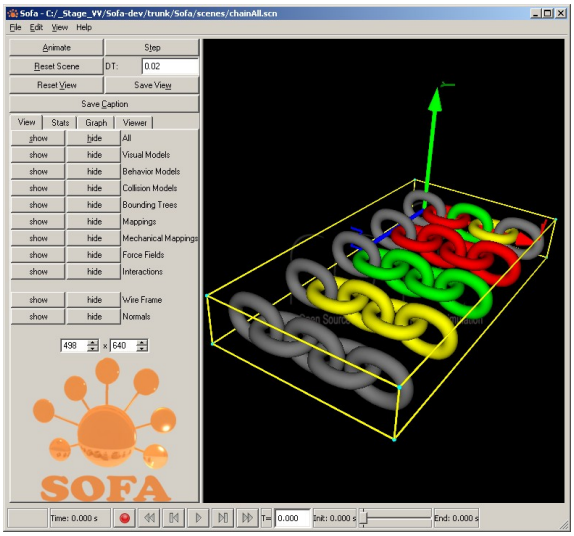


FIG. 4.22 – Le résultat obtenu avec la même scène dans SOFA

A l’issue de cet import, on remarque que les noms des objets (voir figure 4.23) portent bien les mêmes noms que les noeuds du fichier XML SOFA importé. A ceci près que les noms des objets dans Blenders sont uniques. Si tous les maillons de la chaîne de la scène s’appellent “Torus”, dans Blender il seront appelés “Torus”, “Torus.001”, “Torus.002”, etc.

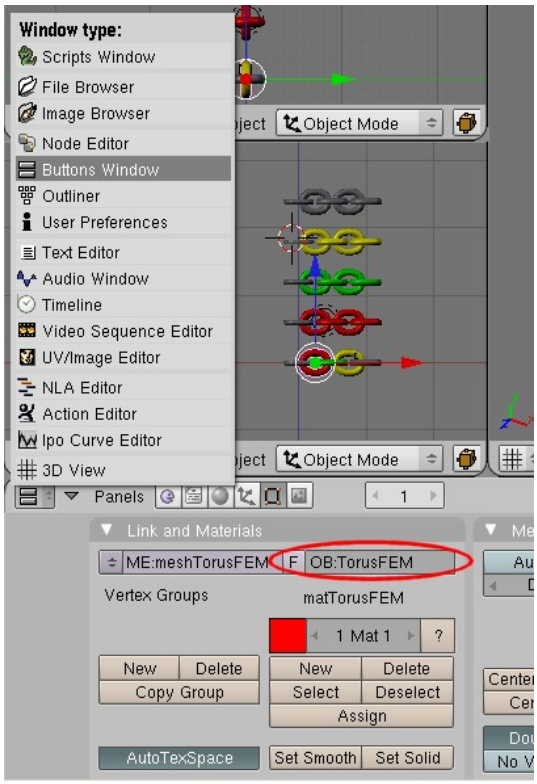


FIG. 4.23 – Nom de l’objet selectionné dans une fenêtre “Button windows”

Description de l’algorithme utilisé

– La fonction d’import parcourt l’arbre du fichier XML SOFA en profondeur (fonction “importer” du fichier “main_CImport_depuis_SOFA.py”).
  – Si un noeud “Node” de l’arbre contient un noeud enfant qui s’appelle “Object” et qui a un attribut “Type” de valeur “MechanicalObject”, alors
    – La valeur de l’attribut “name” du noeud “Node” est mémorisée comme nom du nouvel objet à ajouter dans Blender (je stocke ce nom dans la variable “`strNomObjet`”)

  - La “profondeur” où a été trouvée ce noeud est stockée dans la variable `nNiveauTrouve`
  - Tant que le parcours des noeuds ne descendra pas en dessous de la profondeur `nNiveauTrouve`, tous les nouveaux noeuds parcourus serviront à collecter des informations pour l’objet “`strNomObjet`” trouvé pour Blender
  - La “collecte des informations” se base sur les attributs
    - “dx, dy, dz” et “rx, ry, rz” pour les positions
    - “color” pour la couleur
    - “filename” pour le nom de fichier “.obj”. Par défaut, l’algorithme prend pour valeur de maillage le fichier spécifié par le premier “filename” (en général, c’est le fichier visuel)
    - (voir fonction “def getInformations_depuis_noeud(self, noeud) :” du fichier “sp_Solide.py”)
- Si le parcours arrive en dessous de la profondeur `nNiveauTrouve` ou que le parcours de l’arbre se termine, l’objet “solide” renseigné est ajouté dans la scène courante de Blender (voir méthode “def ajouter_dans_la_scene(self, scene) :” du fichier “sp_Solide.py”).

Organisation des fichiers

Ce script est composé de plusieurs fichiers Python (`.py`). Pour éviter de tous les mettre dans la racine des scripts, je les place dans un dossier qui porte le nom du script, suivi du numéro de version.

C’est le fichier “Import_importer_de_SOFA.py” qui est lié au menu “File \ Import” de SOFA (voir figure 4.24).

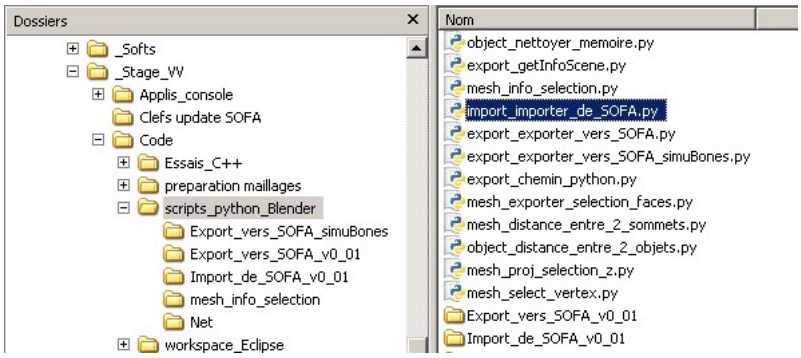


FIG. 4.24 – main du script “Import scènes SOFA”

Ce fichier appelle le dialogue Blender de choix du fichier XML SOFA de scène à importer dans Blender et appelle ensuite la fonction “Importer” du fichier “main_CImport_depuis_SOFA.py” (voir figure 4.25).

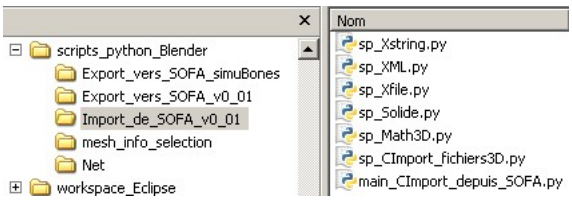


FIG. 4.25 – fichiers du script “Import scènes SOFA”

Le programme se découpe en deux fichiers principaux (lire l’algorithme “Description de l’algorithme utilisé” page 58) :

- Le fichier main_CImport_depuis_SOFA.py (classe “CImport_depuis_SOFA”) qui contient les méthodes :
  - `def __init__( self, strRepertoireRacineAppli ):`
  - `importer` (méthode principale)
  - `def calculer_nom_objet_sans_doublons( self, strNomObjet, liste_des_nom_d_objet ):`
  - `def effacer_la_scene( self ):`
- Le fichier sp_Solide.py (classe “Solide”) qui sert à stocker les informations de l’objet lu dans l’arbre de la scène XML SOFA et à les ajouter dans la scène Blender. Il contient les méthodes suivantes :

- `def initialiser( self ):`
- `def getInformations_depuis_noeud( self, noeud ):`
- `def ajouter_dans_la_scene( self, scene ):`

Les autres fichiers servent de réservoir de petites fonctions utilisaires pour les chaines, les fichiers XML ou les fichiers 3D “.obj” ...

4.4.2 Script “Export des scènes Blender vers SOFA”

Versions des logiciels utilisés

Les scripts suivant ont été écrits avec Blender 2.46 sous Windows XP pro 2002, service pack 2. Ils devraient fonctionner normalement sur toutes les machines (Linux ou Windows) et Blender à partir de la version 2.40, sans compilation.

Utilisation

Cette fonction d’import fonctionne avec un fichier XML d’initialisation (dans le dossier qui contient tous les fichiers source de la fonction d’import). Le fichier XML ci-dessous (listing 4.5) sert juste à ouvrir l’explorateur de fichiers de Blender directement dans le fichier des scènes, sans avoir à naviguer à chaque fois vers les fichiers .scn SOFA.

Listing 4.5 – Fichier XML d’initialisation “Config.xml”

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Config_Export_Sofa xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="C:\_Stage_VV\Code\scripts_python_Blender\
    Export_vers_SOFA_v0_01\config.xsd">
  <Chemins strChemin_modeles_visuels="C:\_Stage_VV\Sofa-dev\trunk\Sofa\scenes\
      VisualModels"
        strChemin_modeles_collision="C:\_Stage_VV\Sofa-dev\trunk\Sofa\scenes\
            CollisionModels"
        strChemin_scene="C:\_Stage_VV\Sofa-dev\trunk\Sofa\scenes"/>
  <Fichiers strNomCompletExecutableSofa = "C:\_Stage_VV\Sofa-dev\trunk\Sofa\bin\
      runSofa.exe"/>
</Config_Export_Sofa>
```

La fonction d’export de Blender vers SOFA exporte les positions, les couleurs et éventuellement (suivant les choix de l’utilisateur) les maillages de la scène.

Les conditions pour réussir un export sont les suivantes :

1. le fichier de scène SOFA doit exister
2. le nom des noeuds de l’arbre de scène XML SOFA et les noms des objets dans la scène Blender doivent correspondre (si les noms ne correspondent pas, les positions de la scène XML ne sont pas mises à jour)

Si ces conditions sont réunies, voici la procédure à utiliser pour “exporter” des scènes Blender dans SOFA :

1. L’accès à la fonction d’export se fait par le menu de Blender “File - Export” (voir figure 4.27)
2. Ensuite on sélectionne le fichier scène SOFA à importer (figure 4.28)
3. On peut alors sélectionner d’exporter les maillages des objets de la scène Blender comme maillage de visualisation et / ou comme maillage de collision (voir 4.29)
4. Si on ouvre la scène avec SOFA, on vérifie que les modifications on été prises en compte 4.30)

Les figures 4.31 à 4.37 présentent des exports de Blender vers une scène SOFA en choisissant d’autres options et d’autres modifications de la scène.

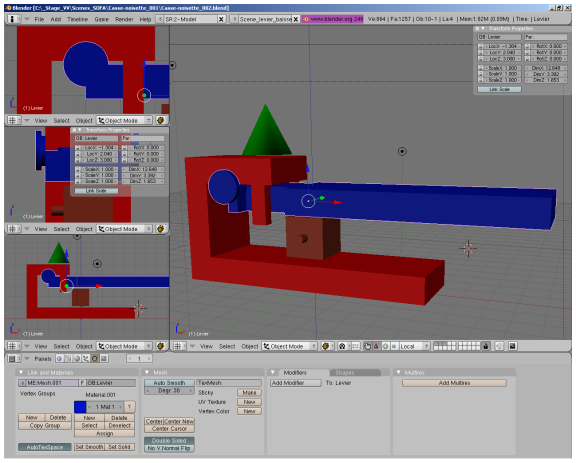


FIG. 4.26 – Scène test dans Blender

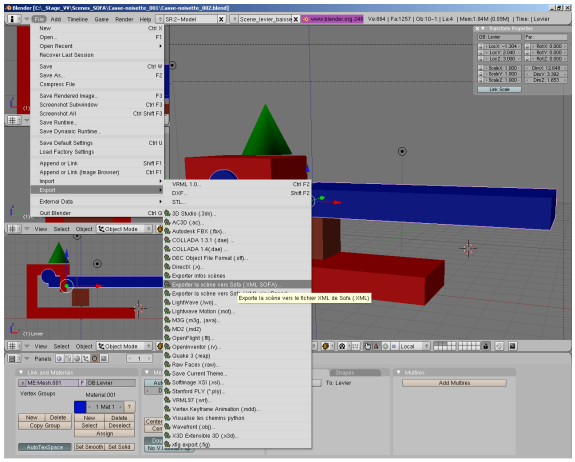


FIG. 4.27 – Appel de la fonction d’export par le menu “File - export”

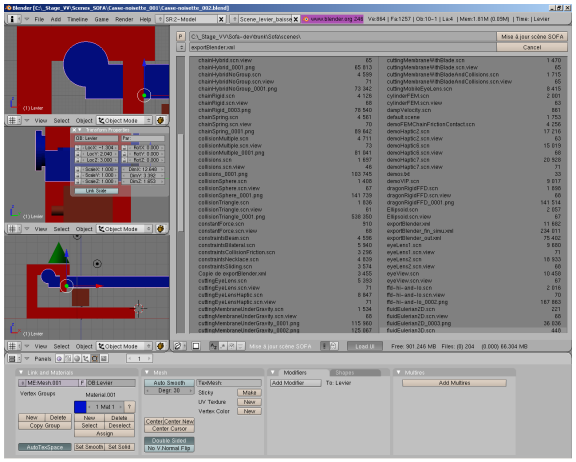


FIG. 4.28 – Choix du fichier scène SOFA à mettre à jour

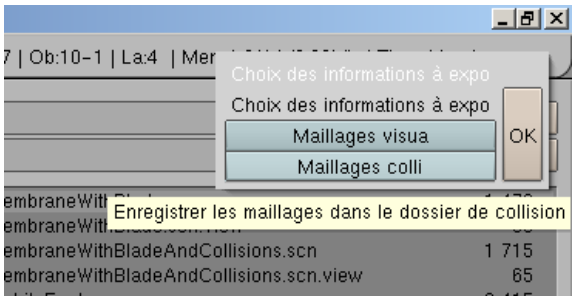


FIG. 4.29 – Options à cocher pour indiquer si on veut mettre à jour les fichiers de maillage de la scène (le maillage de visualisation et le maillage de collision)

La fenêtre de dialogue figure 4.29 correspond au code ci-apès (listing 4.6). On voit que le texte affiché est coupé pour l’instant (au 11 juin 2008). Une solution pour améliorer la lisibilité sera peut-être trouvée d’ici la fin du stage ...

Listing 4.6 – Code Python pour le choix des options d’export

```
    # Récupération de réglages complémentaires
2   EXPORT_FICHIERS_OBJ_VISU = Draw.Create(0)
    EXPORT_FICHIERS_OBJ_COLLISION = Draw.Create(0)
4
    # Récupération des options de l'utilisateur
6   pup_block = []
    pup_block.append('Choix des informations à exporter :')
8   pup_block.append(('Maillages visualisation', EXPORT_FICHIERS_OBJ_VISU, '
        Enregistrer les maillages dans le dossier de visualisation'));
    pup_block.append(('Maillages collision', EXPORT_FICHIERS_OBJ_COLLISION, '
        Enregistrer les maillages dans le dossier de collision'));
10  if not Draw.PupBlock('Choix des informations à exporter', pup_block):
      return
```

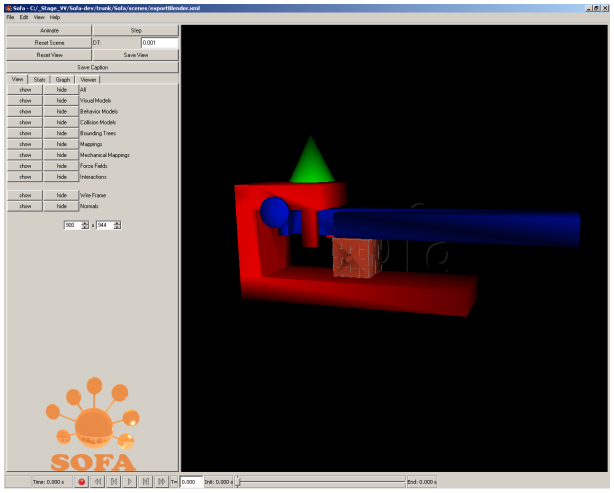


FIG. 4.30 – Scène dans SOFA après export Blender

Dans la figure 4.30 on remarque des ombres qui suivent à peu près le maillage de l’objet. Cela vient du fait que la fonction d’export des maillages pour l’instant (au 11 juin 2008) n’exporte pas les normales dans le fichier .obj (seulement les faces et les sommets).

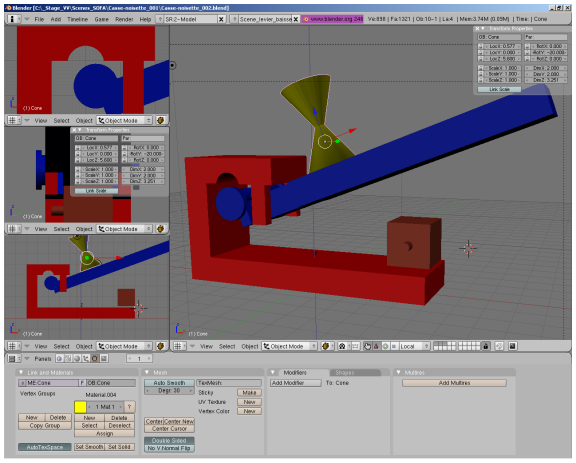


FIG. 4.31 – Modification de la scène dans Blender (modification du maillage “Cone”, changement de sa couleur et des positions des autres objets

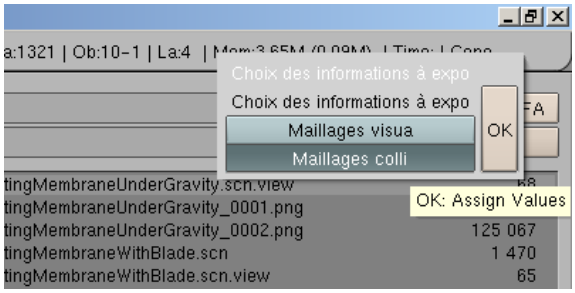


FIG. 4.32 – Choix de ne mettre à jour que le maillage de collision lors de l’export

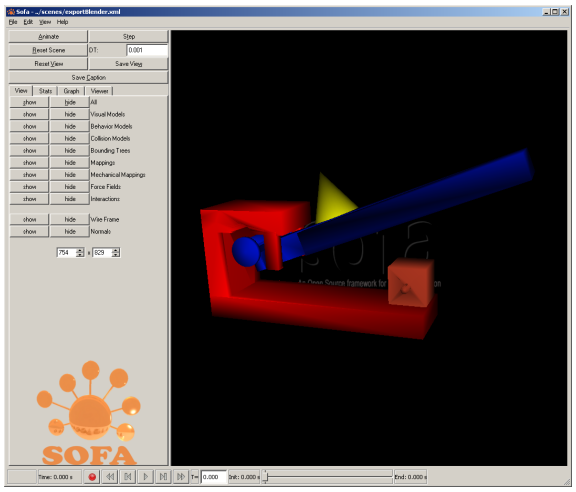


FIG. 4.33 – Résultat dans SOFA, le modèle visuel ne correspondant pas au maillage du cone modifié

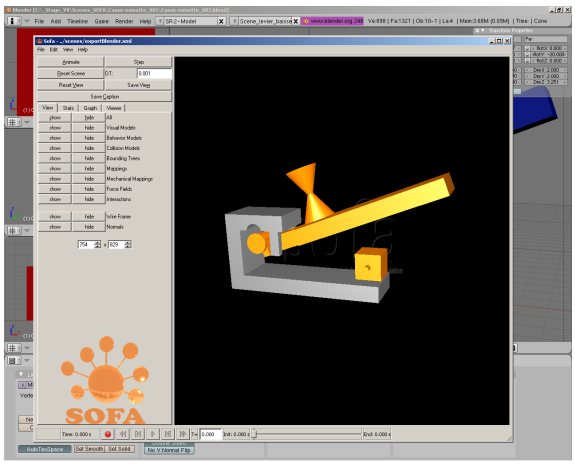


FIG. 4.34 – Résultat dans SOFA, le modèle de collision correspond bien au maillage du cone modifié

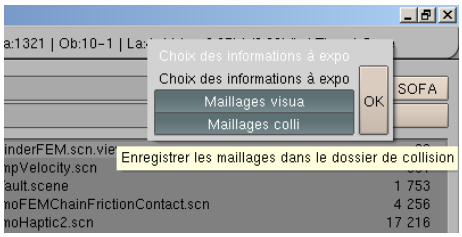


FIG. 4.35 – On exporte de nouveau la scène en cochant les deux boutons de mise à jour des maillages (visuel et collision)...

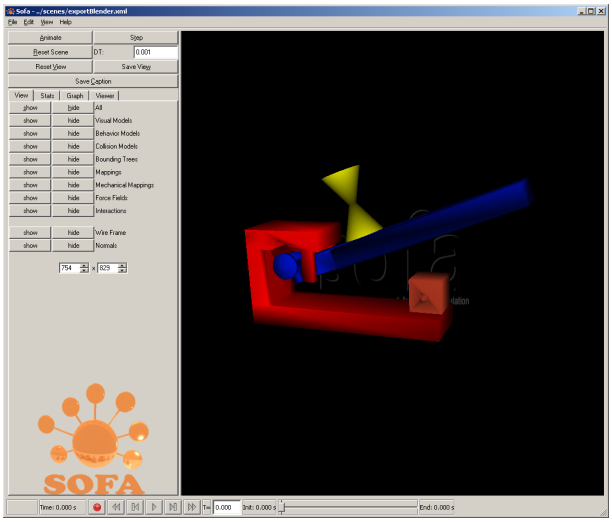


FIG. 4.36 – Le modèle visuel du cone est bien mis à jour dans la Scène dans SOFA après export Blender

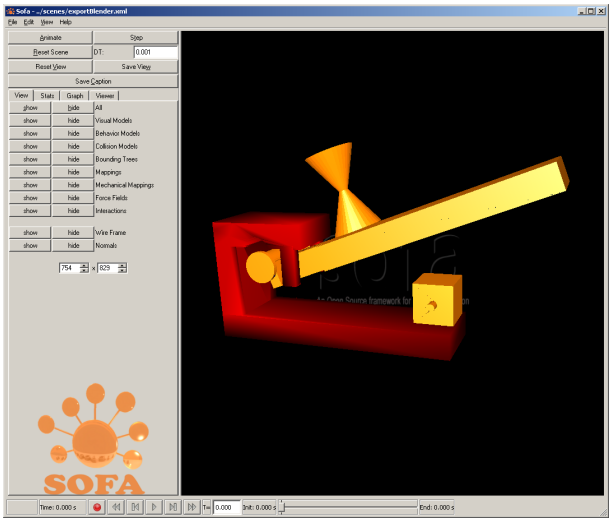


FIG. 4.37 – Le modèle de collision de la Scène dans SOFA après export Blender utilise le même maillage que pour le modèle visuel

Description de l’algorithme utilisé

On parcourt la scène Blender (fonction “exporter” de la classe “CExport_vers_SOFA”) et à chaque objet Blender rencontré :

- on peuple les variables membre d’un “solide” (méthode “Solide : :setParametresObjet”)
- on met à jour le fichier XML (méthode “Solide.mettre_fichier_XML_a_jour()”)
  - La mise à jour du fichier XML consiste à parcourir tous les noeuds du fichier XML à la recherche d’un noeud qui porte le nom de l’objet Blender
    - Si ce noeud n’est pas trouvé, rien ne se passe
    - Si on le trouve, tous les attributs de ses noeuds enfants sont traités par la fonction “Solide.patcherNoeud”

Organisation des fichiers

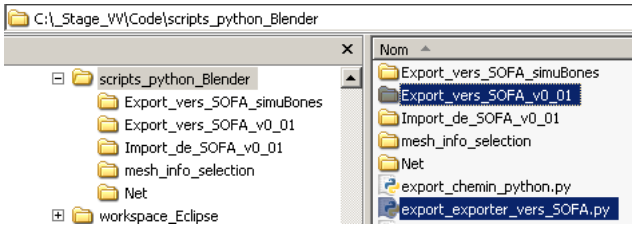


FIG. 4.38 – Le fichier python lié au menu de Blender et le dossier qui contient les autres fichiers Python pour l’export

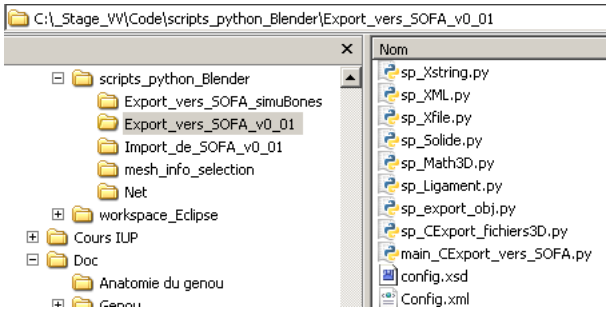


FIG. 4.39 – La liste des fichiers Python de l’export

Comme pour l’import, le programme se découpe en deux fichiers principaux :

1. `main_CExport_vers_SOFA.py`
2. `sp_Solide.py`

– Le fichier main_CExport_vers_SOFA.py (classe “CExport_vers_SOFA”) qui contient les méthodes :
  – `def __init__( self, strRepertoireRacineAppli ):`
  – `def exporter( self, strNomFichierXML ):` (méthode principale)
– Le fichier sp_Solide.py (classe “Solide”) qui sert à stocker les informations de l’objet lu dans la scène Blender et à mettre à jour l’arbre de la scène XML SOFA. Il contient les méthodes suivantes :
  – `def setParametresObjet( self, objet ):` (mise à jour des parametres à partir d’un type “object” de la scène Blender)
  – `def setParametresXML( self, strNomFichierXML ):` (indique quel est le fichier XML à mettre à jour)
  – `def mettre_fichier_XML_a_jour( self, ):` (parcours le fichier XML et met à jour tous les attributs rencontrés)
  – `def patcherNoeud( self, noeud ):` (traitement d’un noeud pour mettre à jour ses attributs)

Les autres fichiers servent de réservoir de petites fonctions utilisaires pour les chaines, les fichiers XML ou les fichiers 3D “.obj” ...

4.5 Obtention des maillages surfaciques réalistes pour le genou

4.5.1 Premier modèle

Le maillage initial des os fournis par François Boux de Casson (voir figure 4.40) a une géométrie qui n'est pas conforme à la description du livre [1] au niveau de la géométrie des glènes.

En effet, le glène interne doit être concave vers le haut suivant un rayon de courbure de 80 mm et le glène externe (celui du premier plan sur la figure 4.40) doit être convexe vers de haut suivant un rayon de courbure de 70 mm.

FIG. 4.40 – Premier maillage de tibia (gauche) fourni par François Boux de Casson

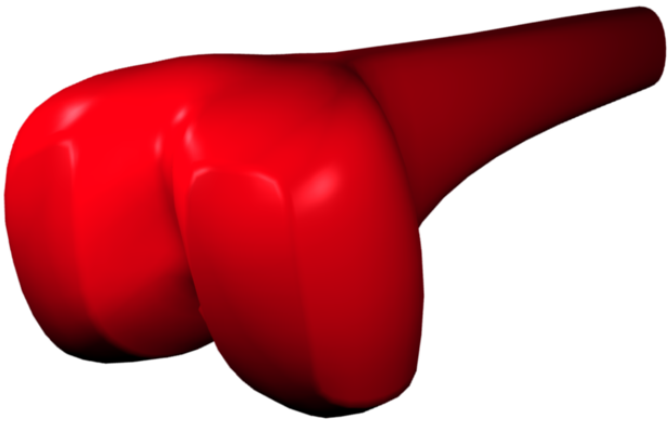


FIG. 4.41 – Premier maillage de fémur (gauche) fourni par François Boux de Casson

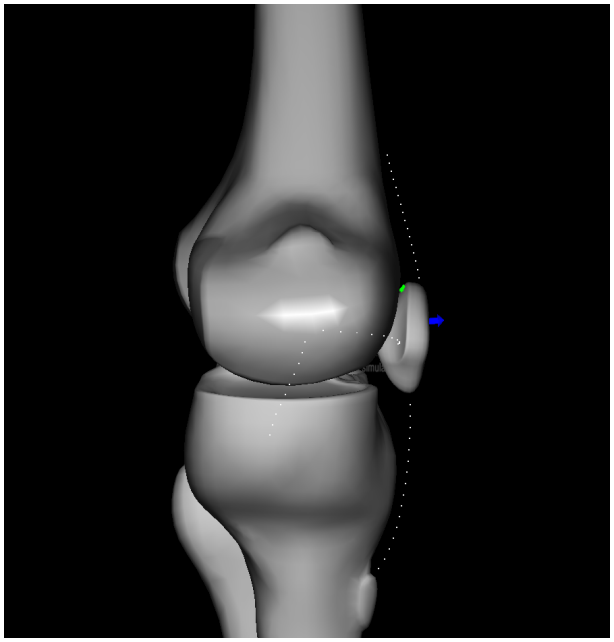


FIG. 4.42 – Premier modèle des os en extension, vue sagittale interne

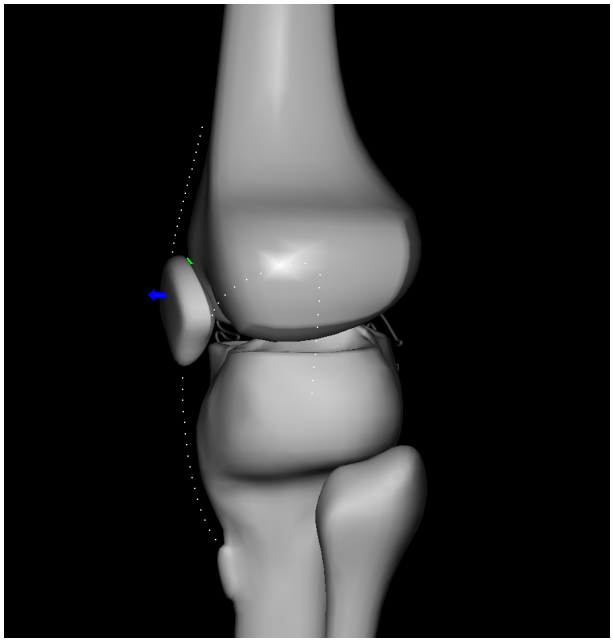


FIG. 4.43 – Premier modèle des os en extension, vue sagittale externe

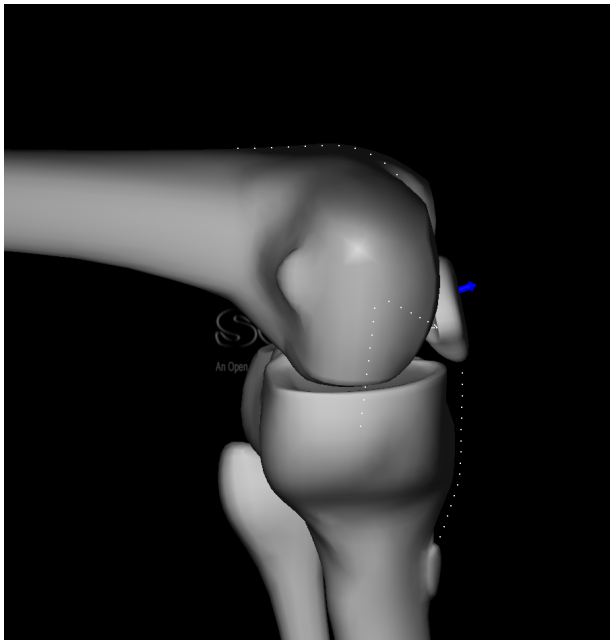


FIG. 4.44 – Premier modèle des os en flexion 90°, vue sagittale interne

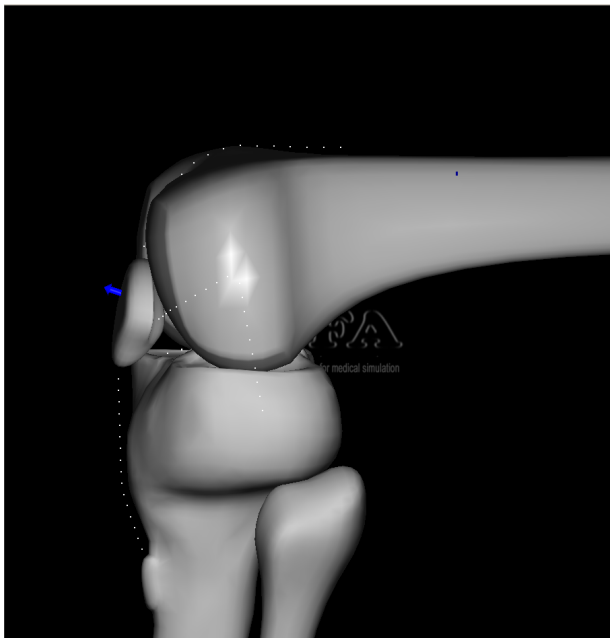


FIG. 4.45 – Premier modèle des os en flexion 90°, vue sagittale externe

Lors des nombreux essais avec ce modèle, avec la position d'attache "normale" des ligaments (en particulier l'attache du ligament latéral externe près de l'extrémité de la fibula[1]), le genou n'avait pas un comportement réaliste. Lors d'un mouvement de flexion par gravité, comme sur les figures 4.45 et 4.44 , autour de la position "fléchie à 90°, les condyles du fémur finissaient leur course à côté du plateau tibial, donnant le comportement complètement irréaliste d'un genou non fonctionnel.

[1] péronné

4.5.2 Modèles obtenus par Imagerie Résonnance Magnétique

Suite aux problèmes géomériques sur les os, Antonin Fontanille (ingénieur SOFA) a été volontaire pour passer un IRM à l'hopital sud de Grenoble (avec l'aide des docteurs Olivier Palombi et Jean-Noël Ravey).

Segmentation des données obtenues

Les données issues du scanner sont des données volumiques au format DICOM. Pour récupérer uniquement les éléments qui nous intéressent (les os), il faut procéder à une segmentation. Effectivement, si on essaye d'extraire le maillage surfacique sans segmentation avec Amira, on obtient les maillages issus des figures 4.46 et 4.47 ci-dessous.

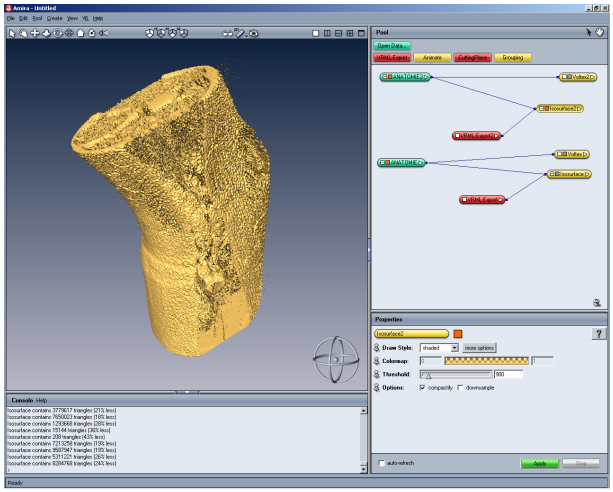


FIG. 4.46 – Tentative d'extraction d'un maillage sans segmentation

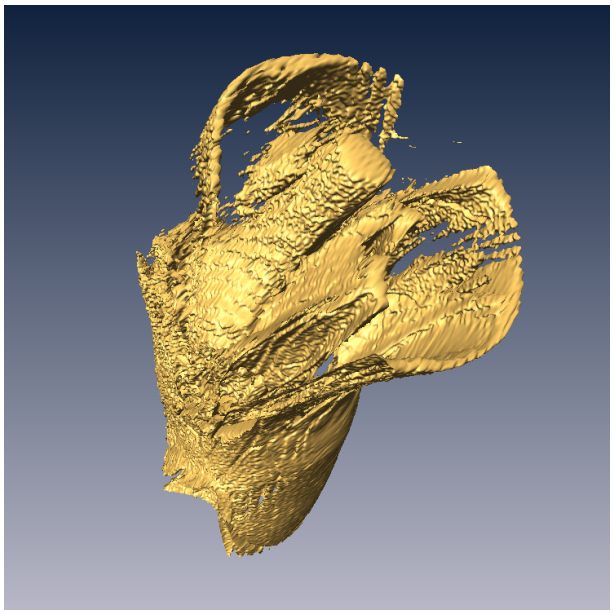


FIG. 4.47 – Tentative d'extraction d'un maillage sans segmentation

La segmentation a été effectuée grâce à la version d'évalutation du logiciel Amira (voir figure 4.48).

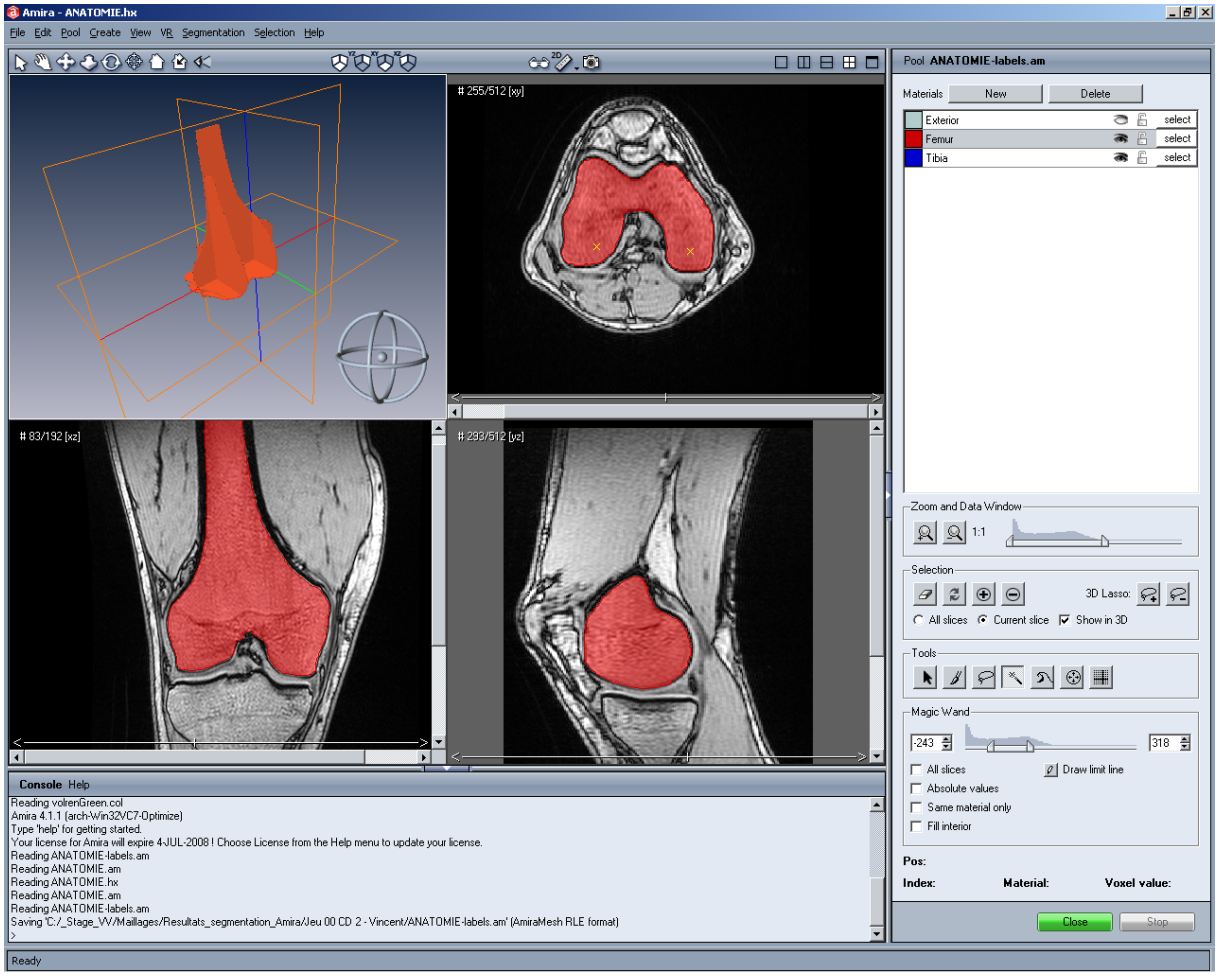


FIG. 4.48 – Vue d'ensemble de la partie segmentation du logiciel Amira

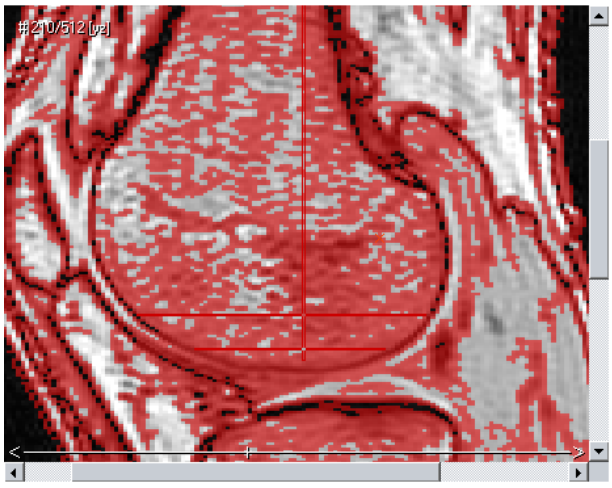


FIG. 4.49 – Sélection de l'os sur un slice de façon automatique

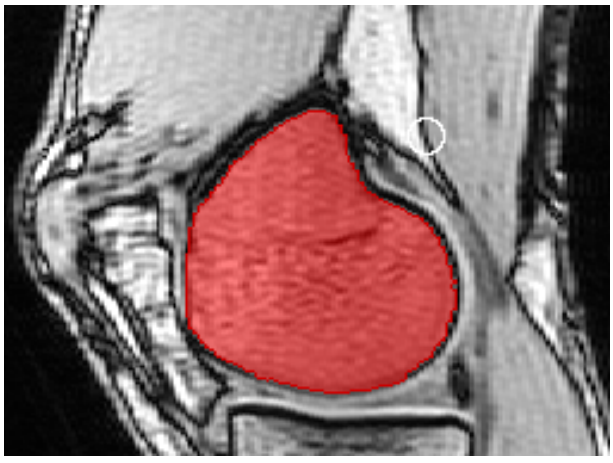


FIG. 4.50 – Sélection manuelle

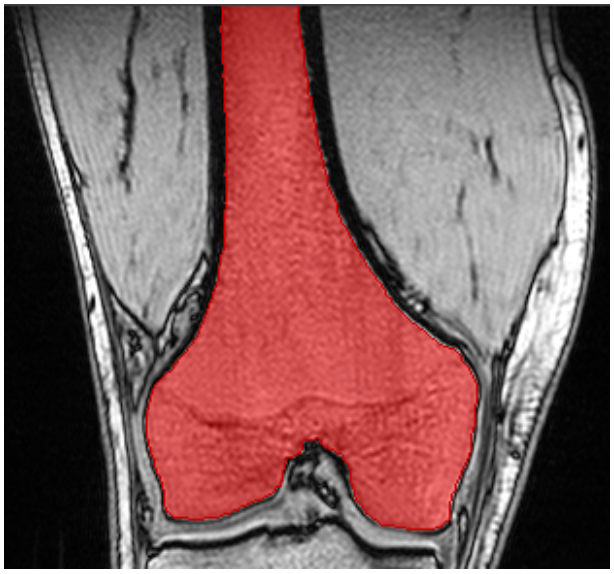


FIG. 4.51 – Sélection manuelle

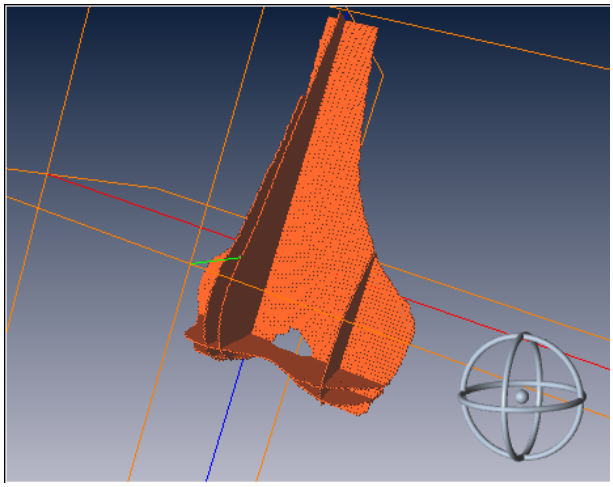


FIG. 4.52 – Affichage instantanné des voxels sélectionnés

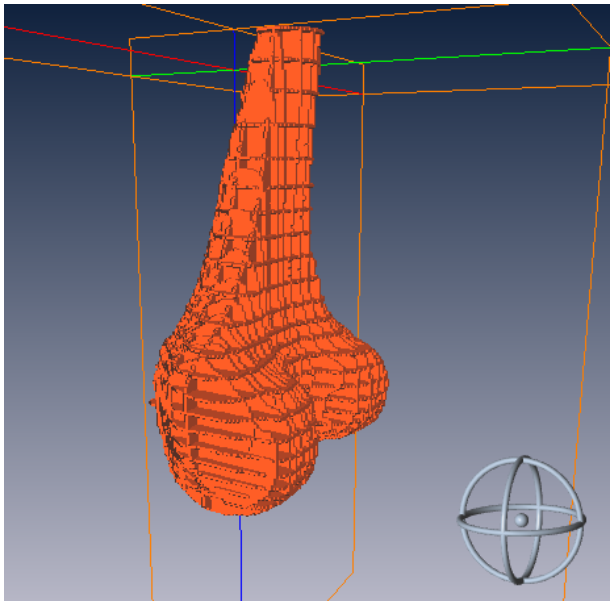


FIG. 4.53 – Sélection finale pour le fémur (jeu de données 000 du CD 2/2)

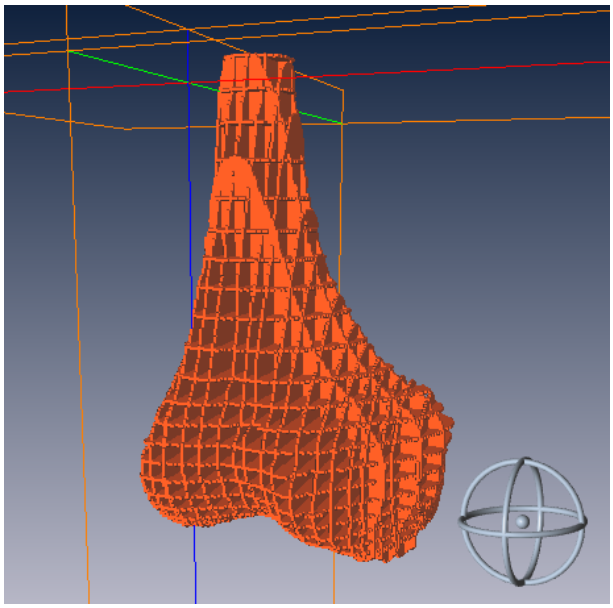


FIG. 4.54 – Sélection finale pour le fémur (jeu de données 000 du CD 2/2)

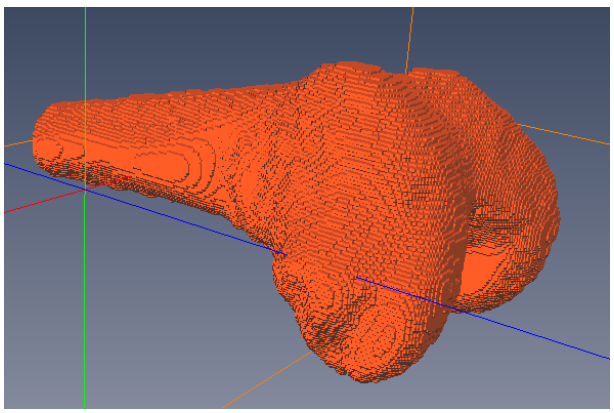


FIG. 4.55 – Nuage de voxel obtenus après interpolation (Wrap)

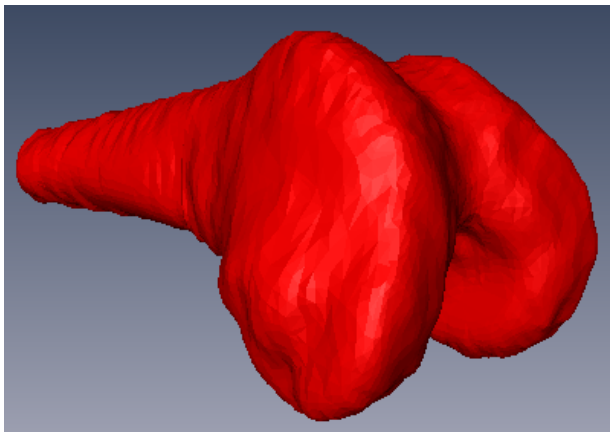


FIG. 4.56 – Triangulation finale (1er essai)

Voici, figures 4.57 et 4.58, le résultat final de ma segmentation (représentation du maillage sans lissage). Les maillages représentent seulement les os. Les contacts par pénalité utilisés avec SOFA créent des

forces de collision avant que les os ne se touchent et jouent finalement le rôle des cartilages.

FIG. 4.57 – Genou IRM, vue gauche

FIG. 4.58 – Genou IRM, vue droite

En septembre (2008), les jeux de données de l'IRM devraient être segmentées par un étudiant du domaine médical et les résultats obtenus seront sûrement meilleurs.

4.5.3 Détermination de la taille physique des maillages exportés par le logiciel Amira

Arima exporte les maillages surfaciques au format VRML 2.0.

Une mesure de référence pour le genou est la largeur du plateau tibial. Pour trouver le lien entre les fichiers 3D exportés par le logiciel Amira, on effectue une mesure au pied à coulisse sur le genou d'Antonin qui a servi à l'IRM.

FIG. 4.59 – Mode opératoire de la mesure

FIG. 4.60 – Hauteur de la mesure

Dans ces conditions, la mesure de la largeur du plateau tibial d'Antonin est exactement de 90.0 mm.

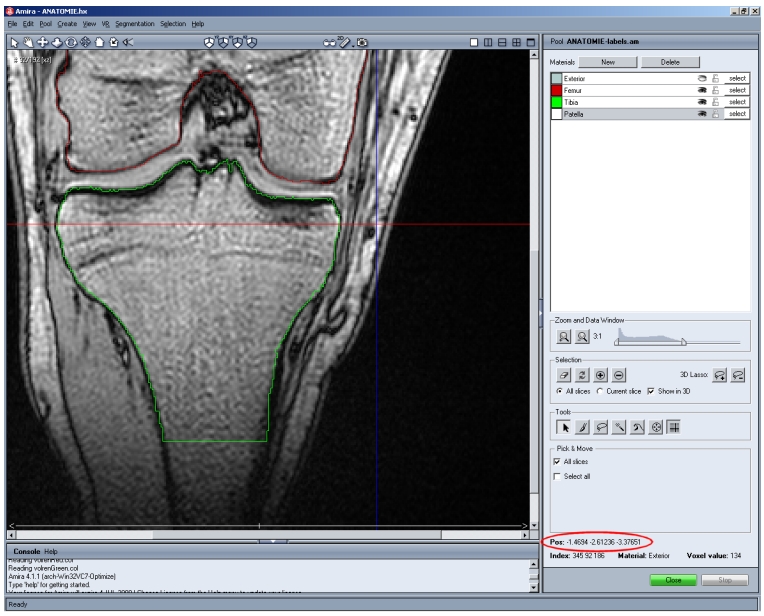


FIG. 4.61 – Mesures dans le logiciel de segmemntation Amira

Dans le logiciel Amira, la position du pointeur de la souris apparaît en bas à droite de la figure 4.61 (cerclé de rouge). La copie d'écran représente le plan xz du genou (x en axe horizontal de l'écran et z en axe vertical).

Sur cette vue la position minimale du plateau tibial (sur la gauche) apparaît à le slice 92/192 avec la valeur -9.91. La position maximale du plateau tibial (sur la droite) est sur le slice 99/192 avec la valeur -2.31. La largeur dans Amira est donc 9.91 - 2.31 = 7.60.

Sur le fichier 3D du tibia exporté par Amira, après recalcul de la position du centre de gravité avec Blender, les extrémas du plateau tibial vont de -3.99 à 3.61, soit une largeur de 3.99 + 3.61 = 7.60 (voir figure 4.62, la mesure de la largeur du plateau tibial à partir d'une sélection de faces).

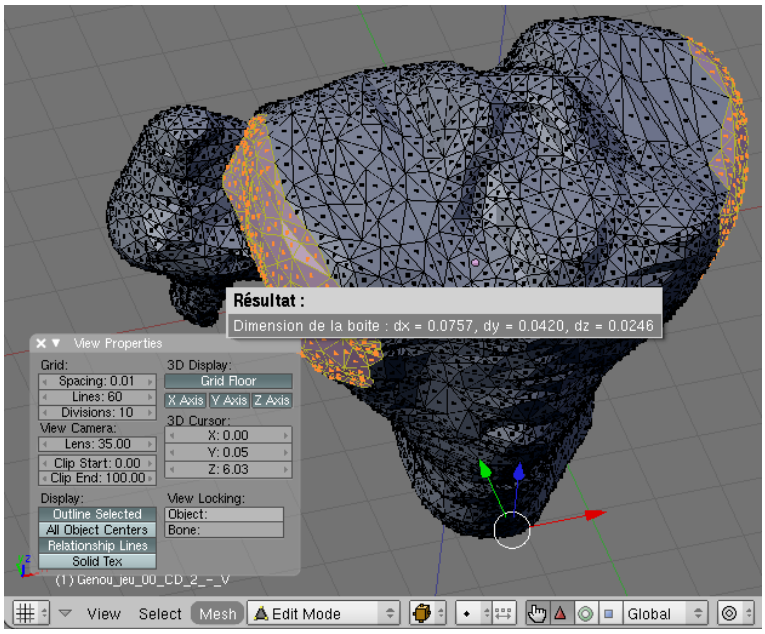


FIG. 4.62 – Mesure de la largeur du plateau tibial avec Blender : 7.57

La position indiquée dans Amira, figure 4.61, est donc de la même grandeur que les données exportées dans le fichier 3D VRML. Il ne reste plus qu'à trouver la grandeur physique de ces valeurs.

J'essaye de retrouver dans Amira la mesure faite avec le pied à coulisse (figure 4.60) sur le genou d'Antonin. C'est une mesure qui a été faite en comprimant la peau. Avec Amira, sur la coupe xz, la peau va de -10.99 à -1.14, soit un écart de 9.85. En prenant une valeur un peu enfoncé sous la peau, pour reproduire la mesure d'Antonin, je note les valeurs -10.47 et -1.42, soit un écart de 9.05.

Le logiciel Amira utilise donc des données en centimètres sur le jeu de données de l'IRM réalisé par l'hopital sud de grenoble.

4.5.4 Choix de l'unité utilisée pour les maillages

Amira exporte donc les données surfaciques du genou d'Antonin (IRM hopital sud) avec le centimètre comme unité de longueur.

Le laboratoire Aesculap utilise des maillages avec des mètres comme unité.

Le logiciel Blender permet facilement de passer des centimètres aux mètres ("scale" 0.01 sur les trois axes).

Le choix se porte donc sur les unités standard : kilogrammes, mètres, secondes.

4.5.5 Obtention de la masse volumique des os

Une recherche sur internet, avec Google, avec les mots-clefs "bone density" ne ma pas permis d'obtenir une réponse claire sur ce sujet.

On peut obtenir la masse volumique des os avec l'expérience suivante :
– peser un os humain
– l'immerger totalement dans un récipient cylindrique (ou à section rectangulaire) rempli d'eau
– mesurer l'écart du liquide sur le bord du récipient

– calculer le volume de l'os par le volume d'eau déplacé et sa masse volumique

En attendant de pouvoir faire l'expérience, sachant que les os ne flottent pas, j'utilise une densité un peu supérieure à celle de l'eau : 1.25, soit une masse volumique de 1250 kg/m^3.

4.6 Calcul des éléments d'inertie des os

Pour obtenir une animation réaliste, il est nécessaire de travailler avec des grandeurs physiques réelles, de calculer la masse, le centre de gravité et les éléments d'inertie des os.

Brian Mirtich a écrit un code en C (volInt.C) qui calcule tous ces éléments en 1995. Son code prend en entrée un fichier au format "Polyhedron" (qui ressemble au format des fichiers OFF ou OBJ).

Pour faciliter l'utilisation de son programme, j'en ai fait un portage en Python pour Blender. Le code fonctionne avec n'importe quel objet de type "Mesh". Il suffit d'entrer au clavier la masse volumique du maillage.

Ce programme tient en un seul fichier Python, il s'appelle "`object_Compute_polyhedral_mass_properties.py`". Pour l'installer, il suffit de le copier dans le répertoire des scripts utilisateurs de Python (voir en annexe "Programmation de scripts Python dans Blender, page 95)

4.6.1 Vérification du bon fonctionnement du programme

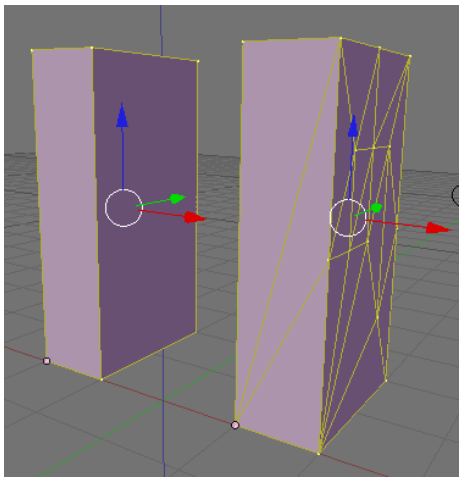


FIG. 4.63 – Deux parallélépipèdes test de même dimension (1x2x4)

Sur la figure 4.63, le maillage à gauche est nommé "Maillage simple" composé de 6 faces rectangulaires et le maillage droit (nommé "Maillage complexe") a une topologie différente, basée sur une triangulation non régulière. Le calcul du centre de gravité et des éléments d'inertie de ces deux maillages doit donner le même résultat.

Voici le résultat du calcul avec une masse volumique égale à 1 :

Elements d'inertie et centre de gravite pour le maillage "Maillage_simple"

```
Masse =            +7.999993
Volume =           +7.999993
Masse volumique = +1.000000

Centre de gravite :  (   +0.500000,   +1.000001,   +1.999999)
```

Matrice d'inertie avec comme origine, le centre de gravité :

$$I = \begin{pmatrix} A = & +13.333311 & -F = & +0.000003 & -E = & -0.000000 \\ -F = & +0.000003 & B = & +11.333316 & -D = & +0.000005 \\ -E = & -0.000000 & -D = & +0.000005 & C = & +3.333326 \end{pmatrix}$$

Elements d'inertie et centre de gravite pour le maillage "Maillage_complexe"

```
Masse =            +7.999992
Volume =           +7.999992
Masse volumique = +1.000000

Centre de gravite :  (   +0.500000,   +0.999999,   +2.000000)
```

Matrice d'inertie avec comme origine, le centre de gravité :

$$I = \begin{pmatrix} A = & +13.333317 & -F = & -0.000000 & -E = & +0.000000 \\ -F = & -0.000000 & B = & +11.333321 & -D = & +0.000001 \\ -E = & +0.000000 & -D = & +0.000001 & C = & +3.333327 \end{pmatrix}$$

$$A = \int_V (y^2 + z^2)\, dm \qquad C = \int_V (x^2 + y^2)\, dm \qquad E = \int_V (x.z)\, dm$$

$$B = \int_V (x^2 + z^2)\, dm \qquad D = \int_V (y.z)\, dm \qquad F = \int_V (x.y)\, dm$$

On vérifie également que le volume et la masse valent $1 \times 2 \times 4 = 8$

Par ailleurs les valeurs des éléments d'inertie sont donnés dans les formulaire de mécanique :

Les formules suivantes sont valables avec $a = 1$, $b = 2$ et $c = 4$, dans le cas des deux parallélépidèdes de test :

$$A = \frac{m}{12} \times (b^2 + c^2) = \frac{8}{12} \times (4 + 16) = 13.3333$$

$$B = \frac{m}{12} \times (a^2 + c^2) = \frac{8}{12} \times (1 + 16) = 11.3333$$

$$C = \frac{m}{12} \times (a^2 + b^2) = \frac{8}{12} \times (1 + 4) = 3.3333$$

D'autre part les valeurs D, E et F doivent être nulles, ce qui est bien le cas.

4.6.2 Exemple d'utilisation sur le maillage d'un fémur

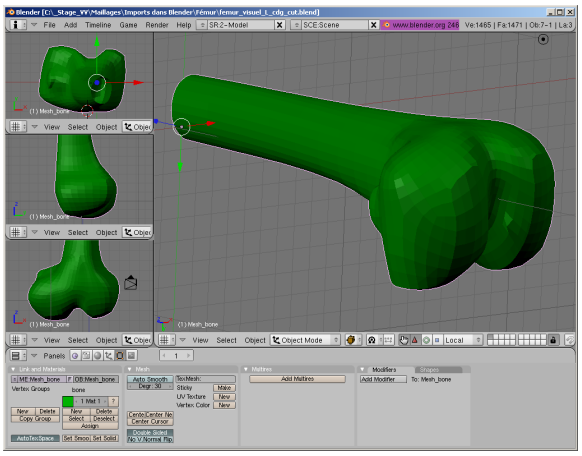


FIG. 4.64 – Le maillage original avec son origine en haut à gauche, dans la fenêtre principale

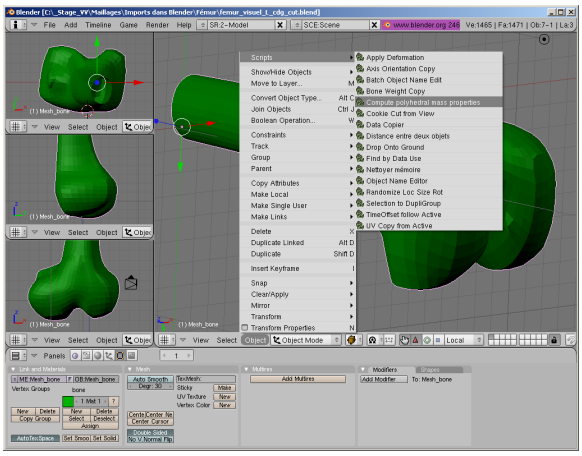


FIG. 4.65 – Appel du script de calcul par le menu "Object"

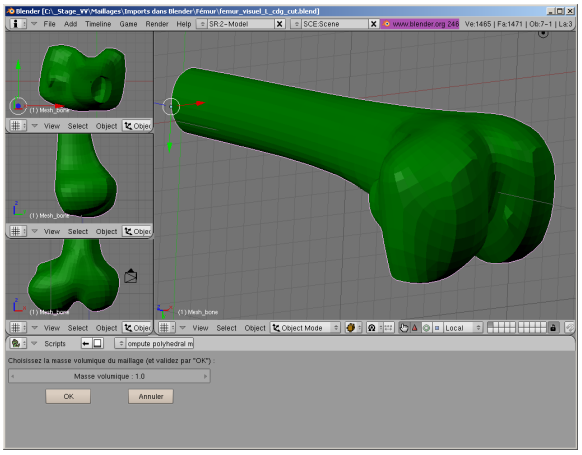


FIG. 4.66 – Choix de la masse volumique dans le dialogue en bas à gauche

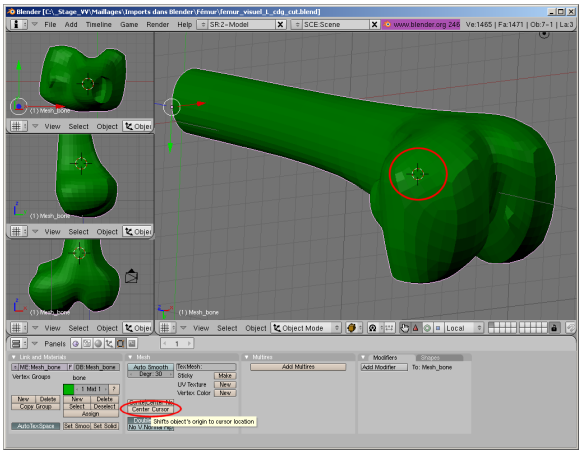


FIG. 4.67 – Le curseur de Blender a été déplacé au centre de gravité du maillage (voir cercle rouge, sur la fenêtre principale)

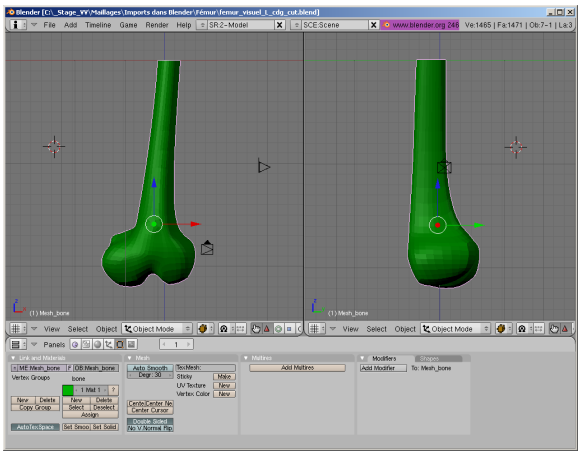


FIG. 4.68 – L'origine du maillage a été déplacée au centre de gravité

```
Elements d'inertie et centre de gravite pour le maillage "Mesh_bone"

Masse =          +323.034402
Volume =         +323.034402
Masse volumique = +1.000000

Centre de gravite :  (   +7.239683,   +2.364255,   -8.008470)

   Matrice d'inertie avec comme origine, le centre de gravité :
```

$$I = \begin{pmatrix} A = & +7103.159067 & -F = & +103.644440 & -E = & -859.320463 \\ -F = & +103.644440 & B = & +7608.784218 & -D = & +393.847155 \\ -E = & -859.320463 & -D = & +393.847155 & C = & +1897.386123 \end{pmatrix}$$

$$\begin{array}{ll} A = \int_V (y^2+z^2)\,dm & D = \int_V (y.z)\,dm \\ B = \int_V (x^2+z^2)\,dm & E = \int_V (x.z)\,dm \\ C = \int_V (x^2+y^2)\,dm & F = \int_V (x.y)\,dm \end{array}$$

FIG. 4.69 – Résultats numériques obtenus dans la console de Blender

4.6.3 Avertissements

Pour que les résultats de cette fonction soient valides, il faut que le maillage soit fermé et que les faces soient toutes orientées dans le même sens.

Avec Blender, pour fermer un maillage, il suffit de sélectionner trois sommets faisant partie de la région à fermer en mode “maillage” et d'appuyer sur la touche “F” (fill) (voir figure 4.71).

Pour orienter toutes les faces vers l'extérieur, toujours avec Blender, on peut utiliser le menu “Mesh - Normals - Recalculate outside” (voir figure 4.70). Pour pouvoir utiliser cette fonctionnalité il faut être en mode maillage (touche TAB) et sélectionner tous les sommets (touche A).

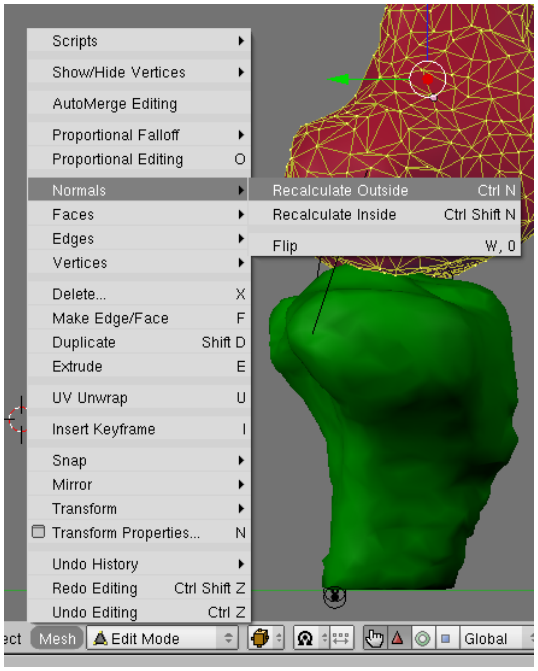


FIG. 4.70 – Orientation des faces

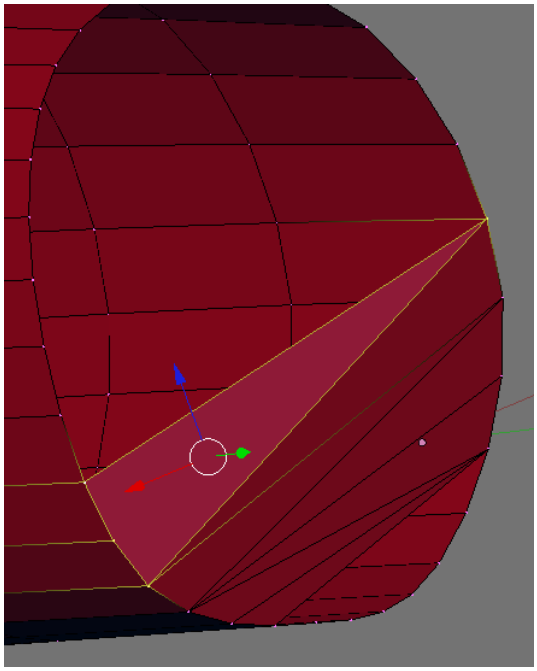


FIG. 4.71 – Fermeture d'un maillage

4.7 Positionnement des éléments de la scène avec Blender

4.7.1 Positionnement du fémur et tibia

Ci-dessous, le placement des os de la scène grâce à Blender et à l'image de référence du genou d'Antonin. Le positionnement a été réalisé pour l'instant par comparaison visuelle par rapport aux vues de la segementation réalisée avec le logiciel Amira.

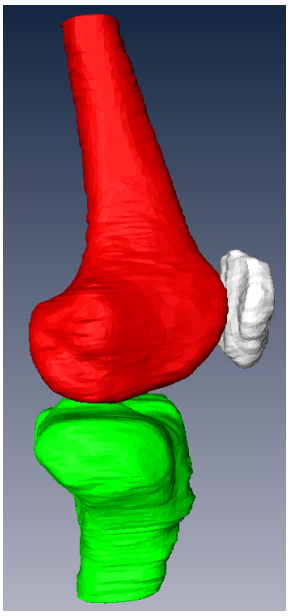


FIG. 4.72 – Vue externe des os avec Amira

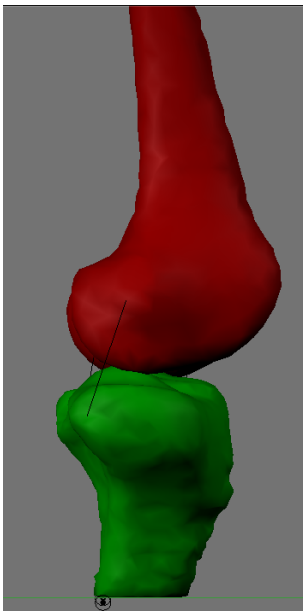


FIG. 4.73 – Positionnement avec Blender

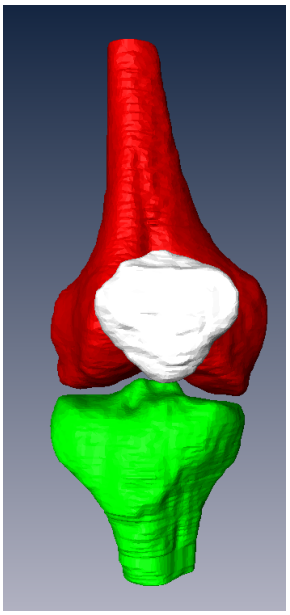


FIG. 4.74 – Vue de face des os avec Amira

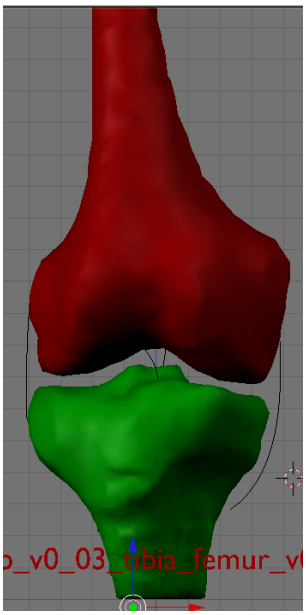


FIG. 4.75 – Positionnement avec Blender

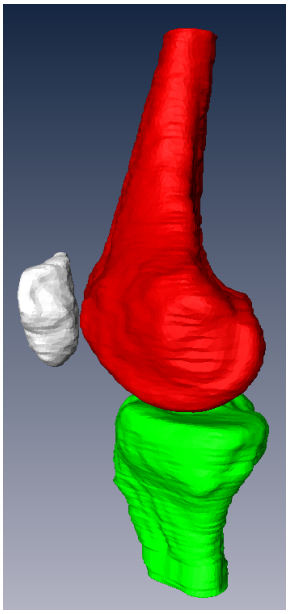


FIG. 4.76 – Vue interne des os avec Amira

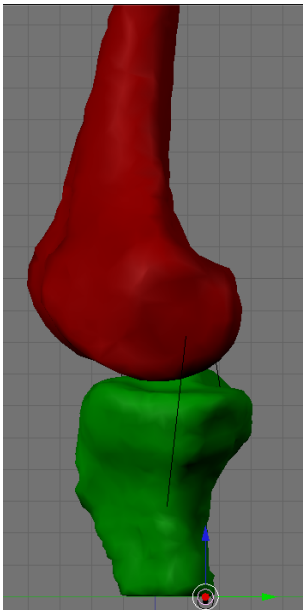


FIG. 4.77 – Positionnement avec Blender

Ce positionnement n'est pas optimum puisque nous disposons à l'origine d'un maillage unique composé de trois parties connexes (tibia, fémur et patella) en situation dans le corps d'Antonin. L'idéal pour le positionnement sera de conserver cette position initiale lors de la séparation des composantes connexes de ce maillage en trois maillages distincts et lors du calcul de leur centre de gravité (voir figures 4.57 et 4.58, page 70).

4.7.2 Positionnement des ligaments

Le positionnement des ligaments s'effectue grâce à la connaissance anatomique de spécialistes du genou, comme Françoix Boux de Casson et Olivier Palombi ou à partir des illustrations du livre de référence sur le genou [1].

Dans la pratique, pour positionner l'extrémité d'une courbe de Bézier (courbe de Bézier qui réprésente la position initiale d'un ligament) sur un noeud du maillage avec Blender, il faut utiliser le curseur de Blender :
– On sélectionne le noeud du maillage de l'os où accrocher le ligament.
– On place le curseur sur cette sélection (SHIFT-S, "snap cursor to selection")
– On sélectionne l'extrémité de la courbe de Bézier à placer sur le curseur
– On place cette extrémité sur le curseur (SHIFT-S, "snap selection to cursor")
Il ne reste ensuite plus qu'à choisir la tangente de la courbe de Bézier pour l'extrémité qui vient d'être placée et à procéder de la même façon pour l'autre extrémité (voir figure 4.79).

Dans le modèle utilisé, un ligament est matérialisé dans sa position initiale par une courbe de Bézier. Or, dans la réalité, les ligaments ont une section conséquente et s'attachent sur des régions plus ou moins grandes, plus ou moins allongées. C'est pour cette raison que sur les maillages des os figures 4.78 et 4.78, ces régions ont été définies (marquage de certaines faces avec d'autres "matériaux").

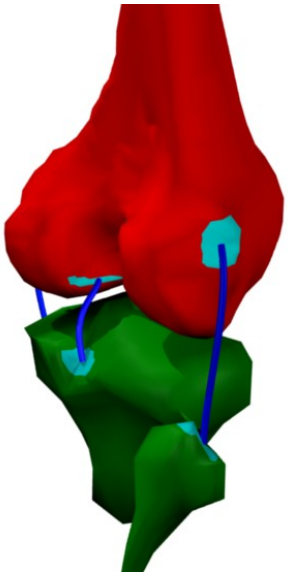


FIG. 4.78 – Vue des zones d'attache (en bleu clair)

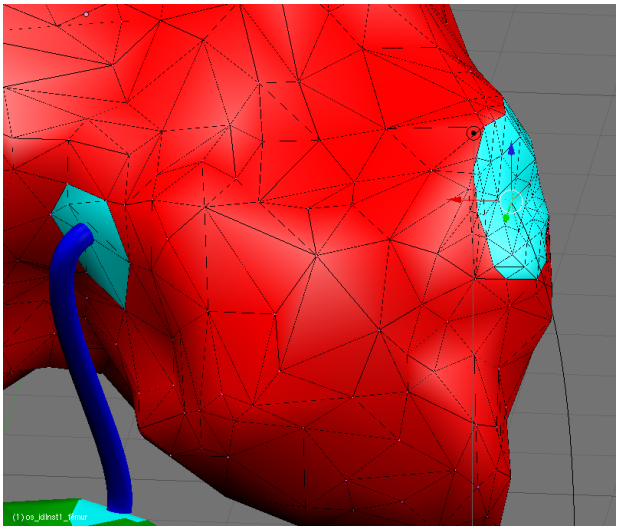


FIG. 4.79 – Détail de l'attache d'un ligament sur un nœud du maillage (à droite)

4.8 Modélisation des ligaments avec SOFA

4.8.1 Modèle masse-ressorts

Le premier modèle utilisé pour les ligaments a été un chapelet de masses-ressorts[2].

Dans SOFA, l'objet correspondant à ce "chapelet" est un `sofa::component::collision::TSphereModel`". L'utilisation de ce composant par code C++ s'effectue avec le code suivant (voir listing 4.7) et correspond à la figure 4.80 ci-après.

Listing 4.7 – "Construction d'un ligament"

```
    // ajout des sphères de collision

    GNode * node_ligament = new GNode;
    strTmp = "node " + (*itLigament).strNom_;
    node_ligament->setName( strTmp );
    node_Ligaments->addChild( node_ligament );

    TSphereModel<Vec3Types> * ligament_spheres;
    ligament_spheres = new TSphereModel<Vec3Types>;
    ligament_spheres->resize( (*itLigament).nNbParticules_ );
    for (int i = 0; i < (*itLigament).nNbParticules_; ++i ) {
      ligament_spheres->setRadius( i, (*itLigament).nRayon_ );
    }
    strTmp = "ligament " + (*itLigament).strNom_;
    ligament_spheres->setName( strTmp );
    node_ligament->addObject( ligament_spheres );

    // ajout de masses

    UniformMass<Vec3Types, double> * masseLig;
    masseLig = new UniformMass<Vec3Types, double>;
    masseLig->setMass( (*itLigament).nMasse_ /
              (*itLigament).nNbParticules_ );
    masseLig->setTotalMass( (*itLigament).nMasse_ );
    strTmp = "masse " + (*itLigament).strNom_;
    masseLig->setName( strTmp );
    node_ligament->addObject( masseLig );

    // -- calcul des positions des attaches sur les os dans le
    //  repère absolu
    // --- recherche des attaches source et destination du ligament
    int nNum_os_source, nNum_attache_source;
    bool bTrouve_source;
    bTrouve_source = cfgScene.getNumerosAttache(
              (*itLigament).strAttacheSource_ ,
              nNum_os_source,
              nNum_attache_source );
    assert( bTrouve_source );

    int nNum_os_destination, nNum_attache_destination;
    bool bTrouve_destination;
    bTrouve_destination = cfgScene.getNumerosAttache(
              (*itLigament).strAttacheDestination_ ,
              nNum_os_destination,
              nNum_attache_destination );
    assert( bTrouve_destination );
```

[2] Ressort + amortisseur

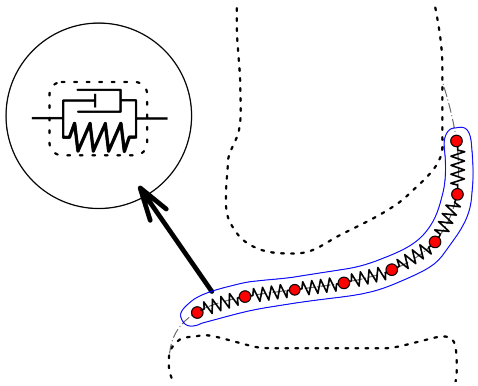


FIG. 4.80 – TSphereModel seul

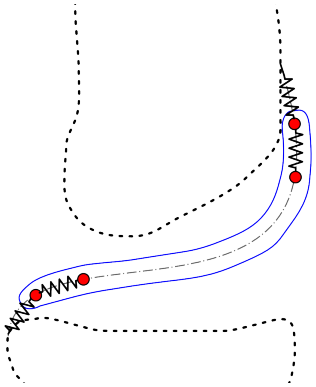


FIG. 4.81 – Ajout de ressorts aux extrémités

Ensuite le ligament est attaché par ses extrémités aux “attaches” placées sur les os par des ressorts-amortisseurs (voir figure 4.81).

Le code C++ SOFA correspondant à l’ajout de ces attaches est dans le listing 4.8.

Les listings 4.7 et 4.8 sont extraits de mon projet “simuBones_v0_03” que l’on peut trouver ici : `\_Stage_VV\Sofa\branches\Sofa-1.0\applications\tutorials\simuBones_v0_03`

(Pour plus de détails sur la localisation du code source des différents projets réalisés, voir la section 4.1 “Organisation du projet”, page 43

Listing 4.8 – ”Ajout de ressorts aux extrémités du ligament”

```
    //
    // -- ajout de ressorts entre l'extrémité "source" du ligament et son attache sur
    //    l'os source
    //
    StiffSpringForceField<Vec3dTypes> * ressort_ligament_os;
    ressort_ligament_os = new StiffSpringForceField<Vec3dTypes>( vAttaches_os[
        nNum_os_source ],
                                    ligament_spheres );
    Os * pOs = cfgScene.getPtrOs( nNum_os_source );
    strTmp = "ressort " + pOs->strNom_ + " - " + (*itLigament).strNom_;
    ressort_ligament_os->setName( strTmp );
    node_ligament->addObject( ressort_ligament_os );

    ressort_ligament_os->addSpring( nNum_attache_source, 0,
                    (*itLigament).nRaideur_,
                    (*itLigament).nAmortisseur_,
                    nLg_entre_particules );
```

4.9 Détail des tables et des données de la base myCorporisFabrica v0.1

C’est l’export LaTeXde mySQL au 30 juillet 2008 de la base “myCorporisFabrica” v0.1 conçue et préremplie par Olivier Palombi (j’ai enlevé quelques données dans le listing ci-dessous pour ne pas surcharger le rapport).

TAB. 4.1: Structure de la table a_faitpartide

Champ	Type	Null	Défaut	Commentaires
id_contenu	int(11)	Oui		
id_contenant	int(11)	Oui		

TAB. 4.2: Contenu de la table a_faitpartide

id_contenu	id_contenant
3	1
13	1
14	1
15	1
20	1
22	1
30	1
31	1
3	2
5	3
6	3
7	3
8	3
9	3
21	4
22	4
23	22
28	22
29	22
53	35
54	35
60	36
62	36

TAB. 4.3: Structure de la table a_sinseresur

Champ	Type	Null	Défaut	Commentaires
id_ea	int(11)	Oui		
id_zone_insertion	int(11)	Oui		

TAB. 4.4: Contenu de la table a_sinseresur

id_ea	id_zone_insertion
53	3
60	3
15	17
54	22
62	22
15	29

TAB. 4.5: Structure de la table acquisition

Champ	Type	Null	Défaut	Commentaires
id_ac	int(11)	Oui	NULL	
Organe	int(11)	Oui		
Type	varchar(30)	Oui		
Date	date	Oui		
Note	mediumtext	Oui		

TAB. 4.6: Contenu de la table acquisition

id_ac	Organe	Type	Date	Note
1	1	IRM	2008-06-01	IRM de genou, jeu de données 00 CD2
2	1	Scanner	1990-07-15	Factice, pour test
3	57	IRM	1990-07-16	Factice, pour test

TAB. 4.7: Structure de la table entite_anatomique

Champ	Type	Null	Défaut	Commentaires
EA_id	int(11)	Oui	NULL	
Nom	varchar(100)	Oui		
Type	int(11)	Oui		

TAB. 4.8: Contenu de la table entite_anatomique

EA_id	Nom	Type
1	genou	5
2	fémur	1
3	épiphyse distale du fémur	1
4	tibia	1
5	fosse intercondylaire du fémur	1
6	condyle médial du fémur	1
7	condyle latéral du fémur	1
8	tubercule supra-condylaire médial du fémur	1
9	tubercule surpa-onduylaire latéral du fémur	1
10	épicondyle latéral du fémur	1
11	épicondyle médial du fémur	1
12	tubercule de l'adducteur	1
13	patella	1
14	tendon du muscle quadriceps fémoral	4
15	ligament patellaire	3
16	base de la patella	1
17	apex de la patella	1
18	bord médial de la patella	1
19	bord latéral de la patella	1
20	fabella latérale	1
21	épiphyse distale du tibia	1
22	épiphyse proximale du tibia	1
23	surface articulaire tibiale supérieur médiale	1
24	surface articulaire tibiale supérieure latérale	1
25	espace inter-articulaire de l'épiphyse proximale du tibia	1
26	éminence intercondylaire	1
27	air intercondylaire ventral	1
28	air intercondylaire dorsal	1
29	tubérosité du tibia	1
30	capsule articualire du genou	4
31	ligament collatéral tibial	3
32	ligament collatéral fibulaire	3
33	ligament poplité arqué	3
34	ligament poplité oblique	3
35	ligament croisé antérieur	3
36	ligament croisé postérieur	3

TAB. 4.8: Contenu de la table entite_anatomique (suite)

EA_id	Nom	Type
37	ménisque latéral du genou	1
38	ménisque médial du genou	1
39	ligament ménisco-fémoral ventral	3
40	ligament ménisco-fémoral dorsal	3
41	rétinaculum patellaire latéral	1
42	faisceau transversal du rétinaculum patellaire latéral	1
43	faisceau longitudinal du rétinaculum patellaire latéral	1
44	rétinaculum patellaire médial	1
45	faisceau transversal du rétinaculum patellaire médial	1
46	faisceau longitudinal du rétinaculum patellaire médial	1
47	tendon du muscle grand adducteur	4
49	corps	5
50	membre pelvien	5
51	insertion proximale	3
52	insertion distale	3
53	insertion proximale	3
54	insertion distale	3
55	insertion proximale	3
56	insertion distale	3
57	hanche	5
58	Cuisse	5
59	membre thoracique	5
60	insertion proximale	3
62	insertion distale	3

TAB. 4.9: Structure de la table instance

Champ	Type	Null	Défaut	Commentaires
Id_ins	int(11)	Oui	NULL	
Id_ac	int(11)	Oui		
id_EA	int(11)	Oui		
X	float	Oui	0	
Y	float	Oui	0	
Z	float	Oui	0	
dX	float	Oui	0	
dY	float	Oui	0	
dZ	float	Oui	0	
Radius	float	Oui	0	
obj_file	varchar(200)	Oui	NULL	

TAB. 4.10: Contenu de la table instance

Id_ins	Id_ac	id_EA	X	Y	Z	dX	dY	dZ	Radius	obj_file
1	1	3	-0.016	-0.015	0.066	0	0	0	0	FemurX.obj
2	1	22	-0.020	-0.008	-0.003	0	0	0	0	TibiaY.obj
4	1	54	-0.006	-0.014	0.020	0.002	0.001	0.006	0	*NULL*
6	1	53	0.015	0.013	-0.033	-0.010	-0.008	-0.004	0	*NULL*
7	1	62	0.008	0.019	0.017	-0.002	-0.006	0.015	0	*NULL*
8	1	60	-0.001	0.014	-0.014	0.002	0.006	-0.015	0	*NULL*

TAB. 4.11: Structure de la table type_anatomique

Champ	Type	Null	Défaut	Commentaires
Type_id	int(11)	Oui	NULL	
Nom	varchar(100)	Oui		
Physique	tinyint(1)	Oui	1	

TAB. 4.12: Contenu de la table type_anatomique

Type_id	Nom	Physique
1	os	0
2	cartilage	0
3	ligament	0
4	tendon	0
5	organe	0

4.10 Procédures d'installation des librairies

4.10.1 Installation de la librairie LibXML2

Procédure d'installation pour DevCpp

DevCpp fonctionne sous Windows avec le compilateur MingW32.

1. Télécharger les fichiers LibXML2 pour Windows sur le site :
   http://www.zlatkovic.com/pub/libxml/
2. Décompresser les fichiers
3. Dans le menu "Projet\Options" du projet de DevCpp mettre à jour les informations suivantes :
   - Onglet "Parametres", ajouter les liens vers les librairies suivantes :
     - .../Libxml2/libxml2-2.6.30+.win32/lib/libxml2.lib
     - .../Libxml2/iconv-1.9.2.win32/lib/iconv.lib
     - .../Libxml2/zlib-1.2.3.win32/lib/zlib.lib
   - Onglet "Répertoires", ajouter les chemins suivants dans le sous-onglet "Répertoire d'inclusion"
     - ...\Libxml2\libxml2-2.6.30+.win32\include
     - ...\Libxml2\iconv-1.9.2.win32\include
   - Dans le répertoire de l'exécutable, ajouter les dll suivantes :
     - libxml2.dll
     - iconv.dll
     - zlib1.dll
4. En-têtes suffisants pour parser les fichiers XML et les vérifier par rapport au schéma :

```
#include <libxml/parser.h>
#include <libxml/xmlschemas.h>
```

5. On peut ensuite utiliser le tutoriel http://julp.developpez.com/c/libxml2/ pour commencer un programme.

Procédure d'installation pour Visual Studio Express

C'est la même procédure que précédemment.
- Les chemins vers les fichiers d'include sont à ajouter dans le menu
  "Projet\Propriétés\Propriétés de configuration\C/C++\Général\Autres répertoires include
- Les librairies ".Lib" sont à ajouter dans le menu
  "Projet\Propriétés\Propriétés de configuration\Editeur de lien\Entrée\Dépendances supplémentaires"

4.11 Installation d'une bibliothèque d'interface graphique

Les principales interfaces graphiques pour Python sont wxWidget, Qt et Thinker. Au niveau interface, Olivier Palombi a déjà développé du code avec Qt et Python pour la base de données "myCorporisFabrica".

De plus Python propose une interface aux bases mySQL très facile à installer et à utiliser ,contrairement à C++ qui réclame d'installer mySQL++ (un wrapper pour l'API C de mySQL) qui est une horreur à compiler. En plus comme j'utilise une installation "légère" de mySQL (easyPhp), il manque des outils mySQL pour compiler mySQL++...

4.11.1 pyQt : Qt pour Python

PyQt permet de lier le langage Python avec Qt pour créer des interfaces graphiques multiplateformes.

4.11.2 Installation

Pour utiliser pyQt il faut installer :

- Python
- Qt
- SIP (http://www.riverbankcomputing.com)
- pyQt (http://www.riverbankcomputing.com)

Avertissement

Certaines extensions pour Windows peuvent parasiter le bon fonctionnement des makefiles. Sur ma machine, par exemple, le "MKS Toolkit" était installé (c'est un logiciel qui reconnaît les commandes Unix sous Windows) et toutes les commandes `copy`, `xcopy`, `if exist ...` des makefiles n'étaient pas reconnues.

Pour désinstaller ce logiciel, désinstaller le logiciel appelé "Unix commands" dans les "Ajout / suppression de programmes" de Windows.

Installation de SIP

SIP est une librairie qui permet de faire des liens entre Python et le C/C++.

Sous Windows, il faut que la variable `PATH` pointe vers
- L'executable Python (`C:\Python25\Python.exe` pour mon cas)
- Le fichier `make.exe` (`C:\Dev-Cpp\bin\make.exe` pour utiliser le compilateur MinGW32 installé avec DevCpp)

Ensuite il faut taper (pour compiler avec MinGW32) :

```
python configure.py -p win32-g++
```

Puis...

```
mingw32-make
mingw32-make install
```

La commande `make install` produit l'erreur suivante :

```
syntax error: got EOF, expecting then
make[1]: *** [install] Error 1
make[1]: Leaving directory 'C:/_Softs/Code/Python/SIP/sip-4.7.6/sipgen'
make: *** [install] Error 2
```

Pour arriver quand même à la fin de l'installation, il suffit d'ouvrir les fichier MakeFile et lire quelles sont les opérations à suivre :
– copier `sip.exe` dans `C:\Python25\Python.exe`
– copier `sip.h` dans `C:\Python25\include`
– copier `syp.pyd` dans `C:\Python25\Lib\site-packages\siplib`
– copier `sip.exe` dans `C:\Python25\Lib\site-packages\sipgen`

Installation de pyQt

Sous Windows, il faut que la variable `PATH` pointe vers
– Les fichiers QT (`C:\QT4.4.0\Bin` pour mon cas)
Pour la compilation avec MingW32, il faut positionner les variables d'environnement suivantes :

```
QMAKESPEC=win32-g++
QTDIR=C:\Qt\4.4.0
```

L'installation de pyQt nécessite l'installation préalable de SIP (voir paragraphe précédent). L'installation de pyQt s'effectue avec l'instruction suivante :

```
python configure.py -p win32-g++
```

Ensuite taper :

```
make
```

puis

```
make install
```

Pour vérifier que l'installation fonctionne, il suffit ensuite d'exécuter ce programme Python :

```
from PyQt4.Qt import *
from sip import *
print SIP_VERSION_STR, QT_VERSION_STR, PYQT_VERSION_STR
```

4.12 Mini tutoriel Blender pour manipuler les scènes SOFA

Il existe de nombreux tutoriels pour Blender sur internet. Les sections qui suivent représentent le minimum à savoir pour configurer Blender et modifier des scènes, c'est un résumé pour ceux qui sont pressés.

Le point le plus important avec Blender est d'appliquer sa règle d'or : "Une main sur le clavier, l'autre sur la souris". Effectivement, si on n'utilise que les menus de Blender à la souris, on n'arrive pas à dessiner.

Le second conseil est de ne pas essayer de faire des scènes trop compliquées tout de suite. C'est le meilleur moyen pour se décourager.

4.12.1 Configuration des fenêtres de Blender

L'écran de Blender est composé d'une mosaïque de fenêtre (par défaut, deux). Chaque fenêtre contient une barre de titre ("Header") (voir repère 1, figure 4.82). Chaque barre de titre contient un bouton déroulant à gauche (voir repère 2, figure 4.82) qui définit le type de fenêtre et les boutons qui seront présents dans la barre de titre.

Chaque barre de titre peut être affichée en bas, en haut de la fenêtre ou masquée (voir figure 4.83). Le menu contextuel apparaît en cliquant avec le bouton droit de la souris.

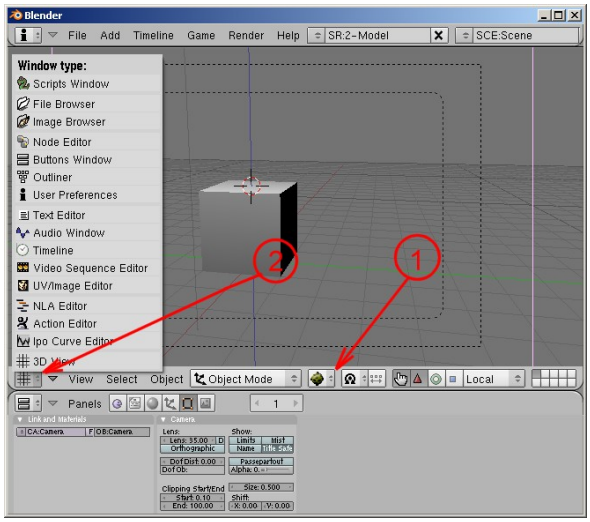


FIG. 4.82 – Composition des fenêtres

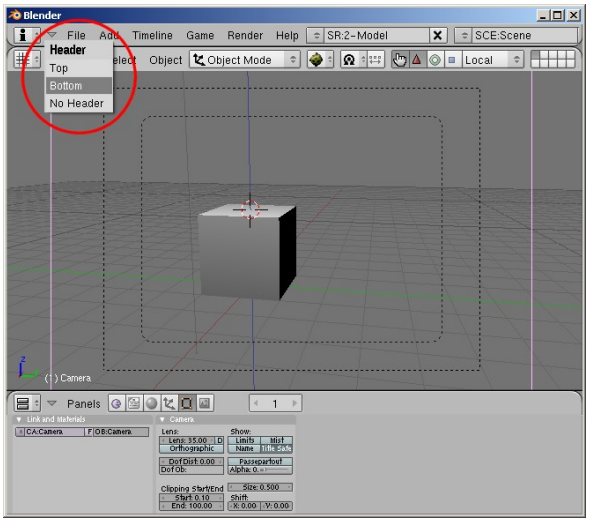


FIG. 4.83 – Choix de la position de la barre de titre

Ensuite, chaque fenêtre peut être coupée en deux ou fusionnée avec sa voisine.

En cliquant avec le bouton droit de la souris, sur la frontière entre deux fenêtres, on obtient le menu suivant (voir figure 4.84). Au final on obtient le résultat figure 4.86.

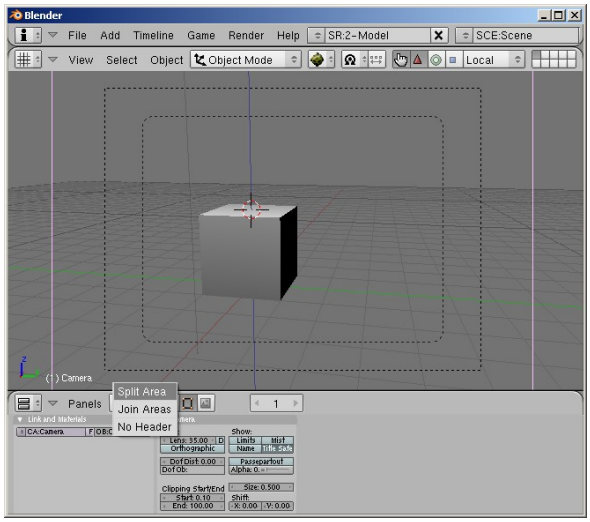


FIG. 4.84 – Composition des fenêtres

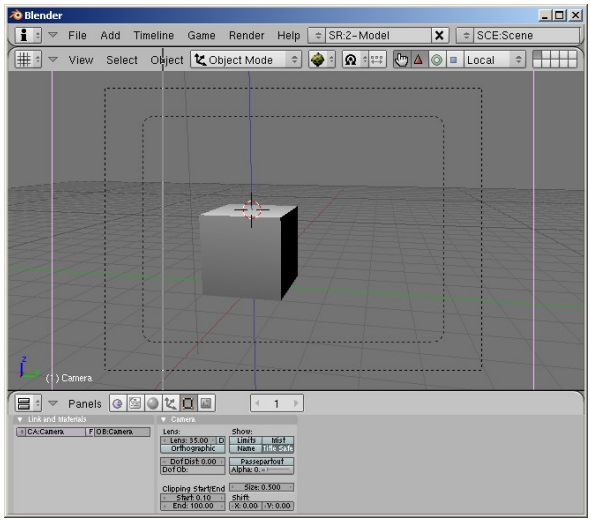


FIG. 4.85 – Choix de la taille de la fenêtre

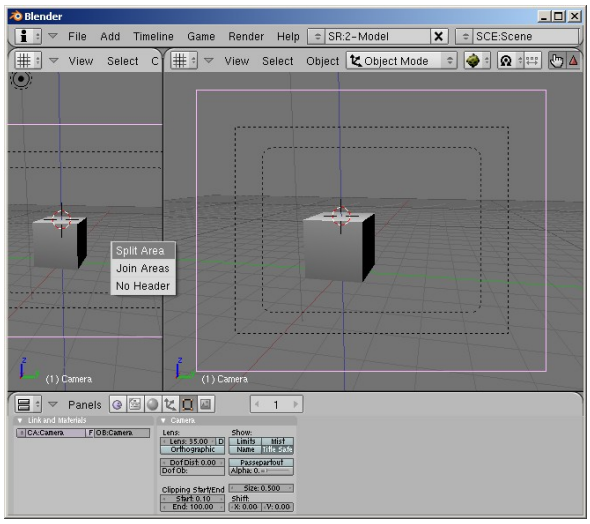


FIG. 4.86 – Résultat obtenu

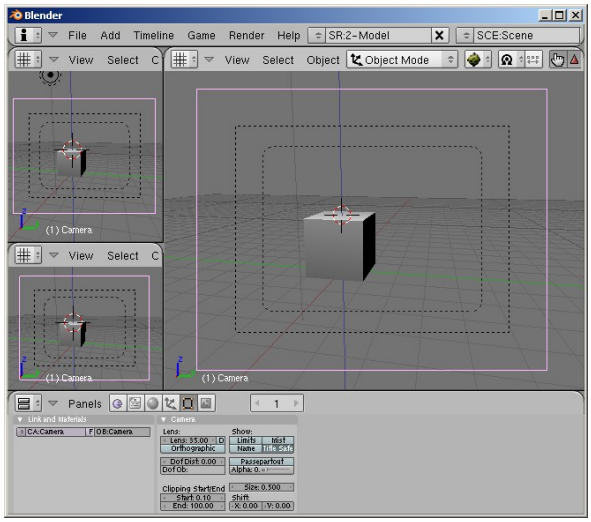


FIG. 4.87 – Nouveau découpage

On peut répéter l'opération indéfiniment (voir figure 4.87).

Maintenant que les fenêtres sont configurées, on peut regarder comment naviguer dans les vues, mais avant, dernier détail : vérifier la configuration de la souris.

4.12.2 Configuration de la souris

Blender utilise souvent des manipulations avec le bouton du milieu de la souris. Avant de continuer, il faut vérifier la configuration à utiliser avant de continuer.

Pour cela, il faut configurer une des fenêtres de Blender en mode "Préférences de l'utilisateur".

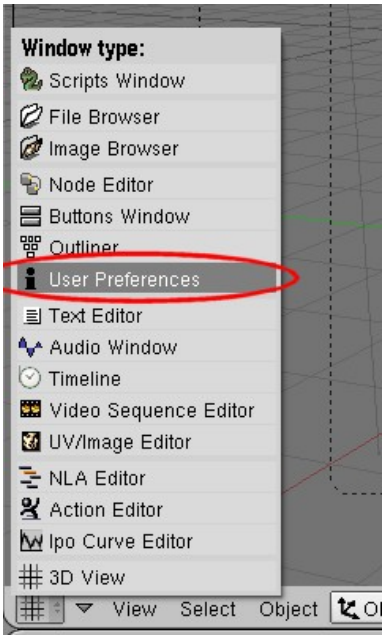


FIG. 4.88 – Accès à la fenêtre des préférences de l'utilisateur

Voici, ci-après l'indicateur de gestion du bouton central de la souris.

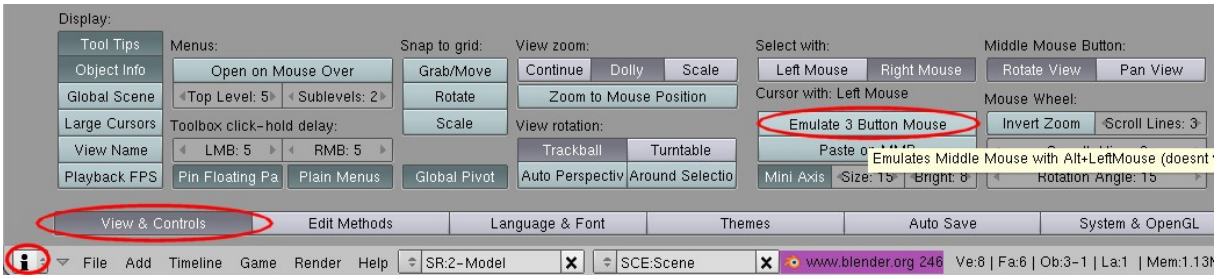


FIG. 4.89 – Configuration de la gestion du bouton central de la souris

4.12.3 Navigation dans les vues

Attention ! Les commandes ci-dessous s'appliquent à la vue dans laquelle est placée le curseur de la souris ! Si la souris est dans une fenêtre qui ne contient que des boutons, il ne se passera rien. Pire, on risque de faire n'importe quoi.

Combinaison de touches	Résultat
clic droit sur un objet	sélectionne l'objet
molette de la souris	zoome ou dézoome la vue active
`shift` + bouton du milieu + déplacement souris	déplace la vue à la souris
bouton du milieu + déplacement souris	rotation de la vue à la souris
`ctrl` + bouton du milieu + déplacement	zoom sur la vue (équivalent de la molette)
touche 5 du pavé numérique	bascule la vue du mode "orthographique au mode perspective"
touche "/" du pavé numérique	bascule la vue de la scène complète à un zoom sur l'objet sélectionné
touche "." (point)	recentre la vue sur l'objet sélectionné
touche 0 du pavé numérique	vue de la caméra active
touche 1 du pavé numérique	vue de face
touche 3 du pavé numérique	vue de droite
touche 7 du pavé numérique	vue de dessus
touches 2, 8 et 4, 6 du pavé num.	rotation de la scène si on n'est pas en mode "vue caméra"
`CTRL` + curseur haut (ou bas)	agrandit ou rétablit la vue sur laquelle se trouve la souris.

NB : La caméra est un objet comme les autres que l'on peut déplacer (voir paragraphe 4.12.4, page 93)

4.12.4 Déplacement d'objets

Combinaison de touches	Résultat
touche `G` (grab)	translation de l'objet sélectionné
touche `R` (rotate)	rotation de l'objet sélectionné
touches `G` puis `X`	translation selon l'axe X
touches `G` puis `Y`	translation selon l'axe Y
touches `G` puis `Z`	translation selon l'axe Z
touches `R` puis `X`	rotation autour de l'axe X
touches `R` puis `Y`	rotation autour de l'axe Y
touches `R` puis `Z`	rotation autour de l'axe Z
touches `G` puis `X - X`	translation selon l'axe X de l'objet
touches `G` puis `Y - Y`	translation selon l'axe Y de l'objet
touches `G` puis `Z - Z`	translation selon l'axe Z de l'objet
touches `R` puis `X - X`	rotation autour de l'axe X de l'objet
touches `R` puis `Y - Y`	rotation autour de l'axe Y de l'objet
touches `R` puis `Z - Z`	rotation autour de l'axe Z de l'objet

Voir l'illustration des repères locaux et globaux pour un objet par les figures 4.90 et 4.91. Sur la figure 4.91, on voit le menu qui permet de passer d'un mode à l'autre.

Sur la figure 4.90, on voit les lignes du repère global (posées sur la grille : rouge = axe X, vert = axe Y, bleu, axe Z).

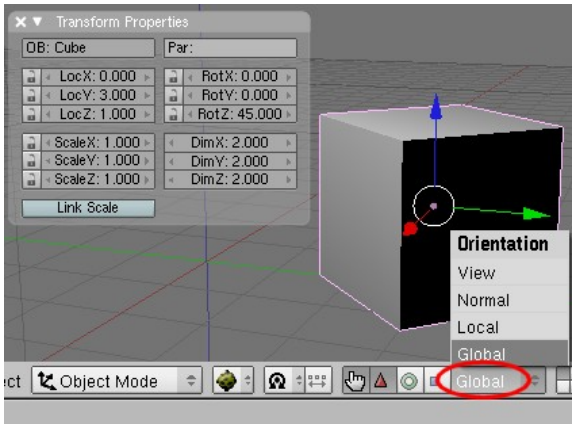


FIG. 4.90 – Cube dans repère global

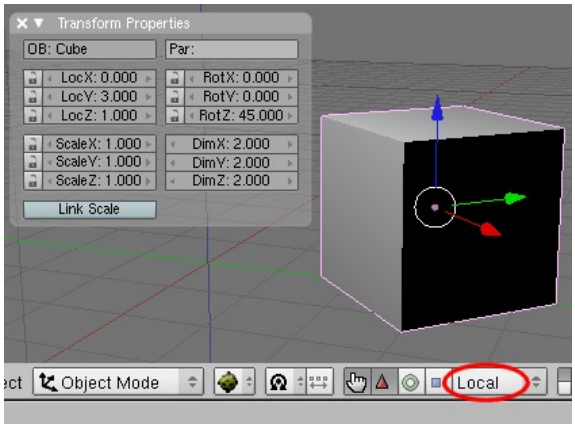


FIG. 4.91 – Cube dans repère local

Les déplacements peuvent s'effectuer à la souris en utilisant ou non les touches SHIFT et CTRL.
- La touche `SHIFT` permet de démultiplier le déplacement de la souris et de placer précisément les objets
- La touche `CTRL` active le mode "accrochage".
  - Par défaut (voir figure 4.92, si la touche `CRTL` est enfoncée, l'objet déplacé se cale sur la grille en translation et tous les 5 degrés en rotation
  - On peut aussi choisir un autre mode d'accrochage (voir figure 4.93), selon les besoins

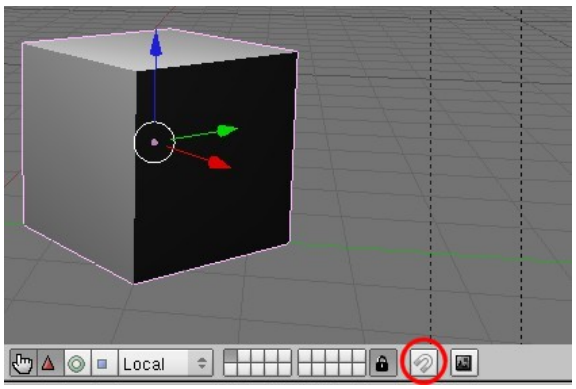


FIG. 4.92 – Accrochage sur la grille

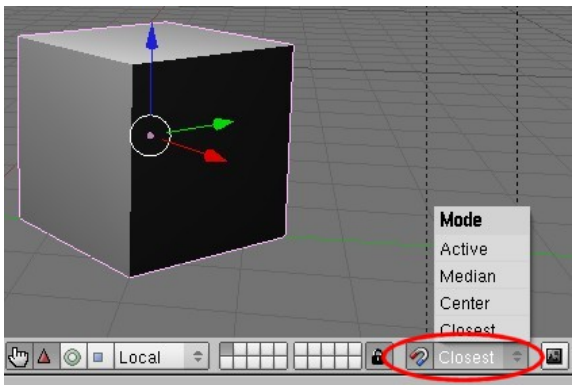


FIG. 4.93 – Choix d'un mode d'accrochage

4.12.5 Annulation des déplacements

Combinaison de touches	Résultat
`U` en mode `edit`	crée un `Undo`
`CTRL Z` en mode objet	annule la dernière opération
`alt` + G	annule un déplacement (Grab)
`alt` + R	annule une rotation

4.12.6 Edition des objets (courbes de Bézier, maillages, etc.)

Combinaison de touches	Résultat
`tab`	Entre ou sort du mode édition
`W`	affiche la boîte à outils

4.13 Programmation de scripts Python dans Blender

4.13.1 Installation

Blender nécessite l'installation au minimum de Python, mais on bénéficie également dans les scripts Blender, de toutes les librairies additionnelles que l'on installe pour Python. On trouve l'installation de la dernière version de Python ici : http://www.python.org/

L'installation de Python par défaut permet de parser des fichiers XML, mais pas de faire de calcul matriciel. Pour avoir accès aux matrices et leurs fonctions de base, il faut installer NumPy. NumPy donne accès aux matrices et vecteurs.

Pour installer toutes les fonctions d'algèbre linéaire, traitement du signal, statistiques, il faut installer Scipy.

Les dernières version de NumPy et SciPy se trouvent ici : http://www.scipy.org/

(Ce site contient également une excellente documentation sur ces librairies)

4.13.2 Quel éditeur utiliser pour Python ?

Il existe le plugin PyDev pour Eclipse, qui fonctionne très bien. Malheureusement, je n'ai pas réussi à l'utiliser avec les librairies Blender. La question est actuellement sans réponse sur le forum www.developpez.com. Je l'ai utilisé pour réaliser le programme d'export des scènes de la base myCorporisFabrica avec pyQt.

Il existe un énorme choix d'autres éditeurs ici : http://wiki.python.org/

4.13.3 Comment écrire et exécuter un premier programme Python ?

Sans Blender

Python est un langage plus simple que le C++ (mais néanmoins objet) qui ressemble un peu au C. Voici un exemple très simple de programme Python :

Listing 4.9 – Exemple simple de programme Python "C :\tmp\Hello.py"

```
1 print "Hello"
a = 1.5
3 b = 1.2
print "a + b = %.3f"%( a + b )
```

Dans mon cas, l'installation de Python est dans le dossier `C:\Python25`.

Pour exécuter le programme précédente "Hello.py", il suffit de taper dans une console :

```
C:\tmp>c:\python25\python hello.py
```

Résultat :

```
Hello
a + b = 2.700
```

Avec Blender

Dans Blender, l'éditeur de texte permet d'écrire et d'exécuter des scripts.

FIG. 4.94 – Configuration d'une fenêtre Blender en éditeur de texte

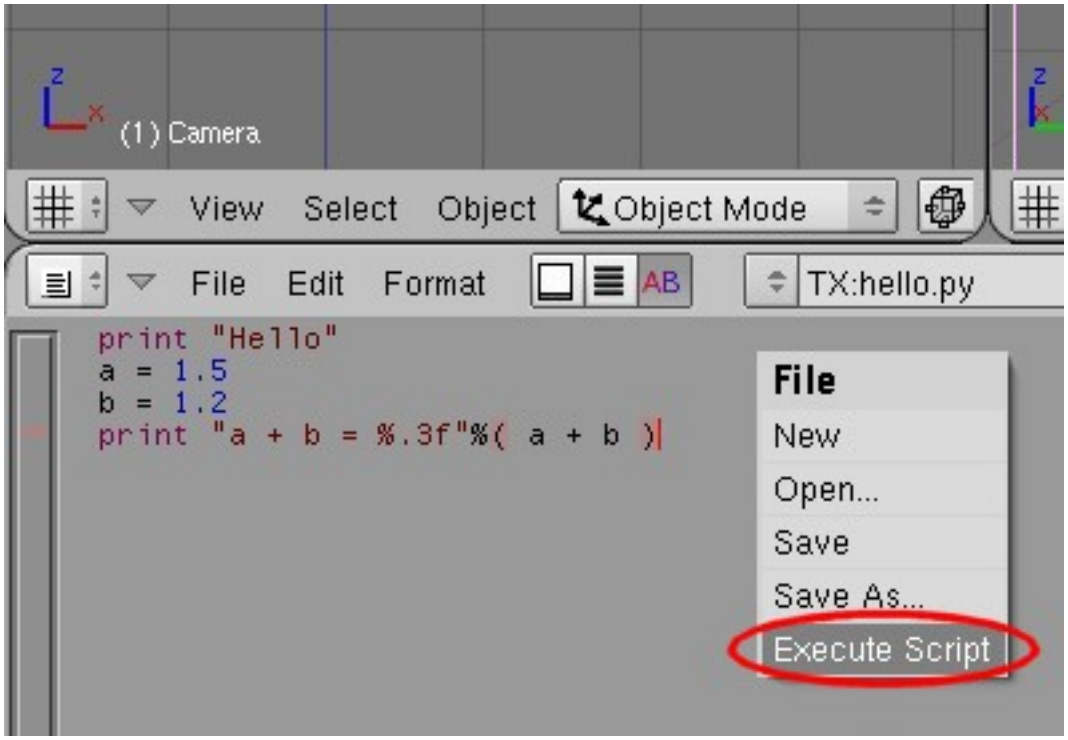


FIG. 4.95 – Exécution d'un script dans Blender

Le résultat suivant s'affiche dans la console de Blender :

```
Hello
a + b = 2.700
```

Liaison de scripts Python avec les menus de Blender

Dans Blender, les scripts Python sont signalés par un petit logo vert représentant un python (le serpent). On en trouve dans les menus `File \ Import`, `File \Export`, `Object`, `Mesh` (voir copies d'écran 4.96 à 4.98.

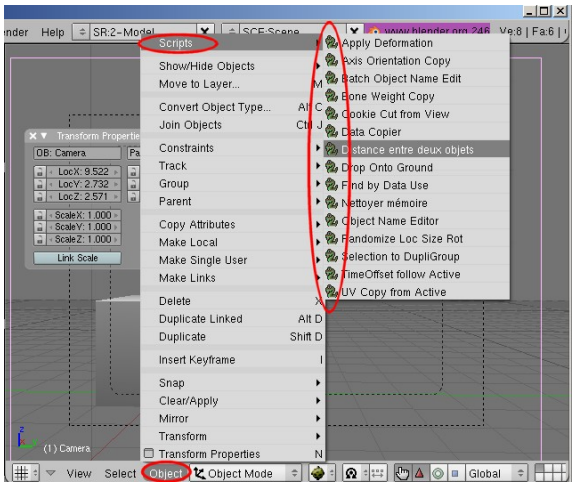


FIG. 4.96 – Scripts dans le menu "object"

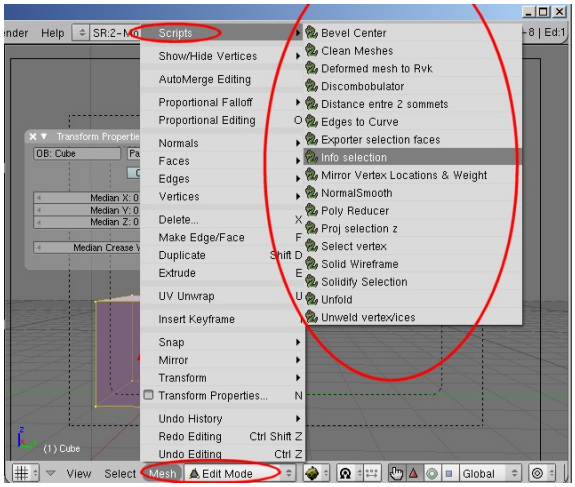
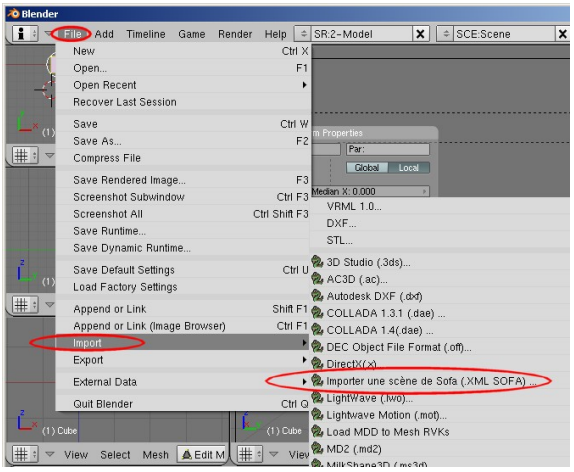


FIG. 4.97 – Scripts dans le menu "mesh"



Dans la figure ci-contre (4.98), on remarque le script d'import des scènes SOFA, entouré en rouge.

FIG. 4.98 – Scripts dans le menu "File - import"

Sous Windows, tous les scripts livrés par défaut avec Blender sont dans le chemin :

`C:\Program Files\Blender\.blender\scripts`

Chaque script a l'extension `.py`. Pour savoir à quel menu il se rapporte, il suffit de l'ouvrir (en général, le nom du fichier donne déjà l'indication). Voici par exemple le début du script "`import_obj.py`" :

Listing 4.10 – exemple du script "import_obj.py" livré avec Blender

```
#!BPY

"""
Name: 'Wavefront (.obj)...'
Blender: 242
Group: 'Import'
Tooltip: 'Load a Wavefront OBJ File, Shift: batch import all dir.'
"""

__author__= "Campbell Barton", "Jiri Hnidek"
__url__= ["blender.org", "blenderartists.org"]
__version__= "2.0"
```

```
14 __bpydoc__= """\
   This script imports a Wavefront OBJ files to Blender.
16
   Usage:
18 Run this script from "File->Import" menu and then load the desired OBJ file.
   Note, This loads mesh objects and materials only, nurbs and curves are not
       supported.
20 """
```

On remarque ligne 6 du listing 4.10 (`Group: 'Import'`), ci-dessus, que ce script est rattaché au menu "`File Import`".

Pour ajouter un script dans un menu Blender, il suffit donc d'ajouter un fichier ".py" avec le "bon" entête dans le dossier des scripts Python de Blender. Cela fonctionne, mais ce n'est pas la bonne solution.

Pour éviter de mélanger ses scripts avec ceux de Blender, il faut configurer l'endroit où sont les scripts utilisateurs dans Blender. Pour cela, ouvrir un une fenêtre Blender en mode "Préférences utilisateur" :

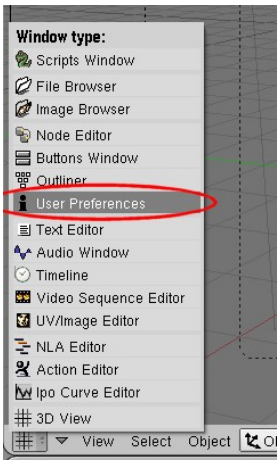


FIG. 4.99 – Accès à la fenêtre des préférences de l'utilisateur

Dans le menu "File Path", configurer le chemin des scripts de l'utilisateur (voir figure 4.100).

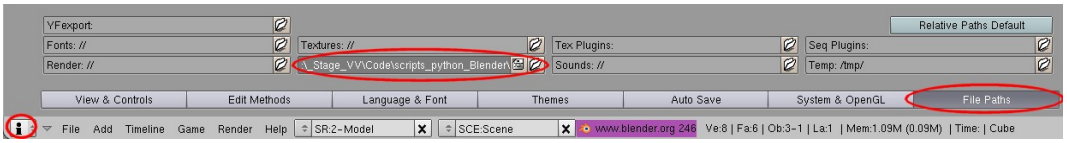


FIG. 4.100 – Configuration des scripts de l'utilisateur

Et donc mes scripts pour Blenders sont isolés dans le chemin "`C:\_Stage_VV\Code\scripts_python_Blender`".

Enfin, pour que Blender se souvienne de ce choix, il faut enregistrer ce changement dans les préférences utilisateur en tapant "`CTRL-U`".

Modification des chemins configurés dans l'environnement Python

En C/C++, on utilise la fonction `#include <...>` pour inclure d'autres fichiers sources dans un projet. En Python, il faut utiliser `import nom_fichier`.

Dans ce cas, il faut que le fichier `nom_fichier.py` soit présent dans la liste des chemins définis dans l'environnement Python.

Pour connaître les chemins configurés par défaut dans Python 2.5, il suffit de taper le code suivant :

```
import sys
print sys.path
```

```
['C:\\Python25\\Lib\\idlelib',
 'C:\\WINDOWS\\system32\\python25.zip',
 'C:\\Python25\\DLLs',
 'C:\\Python25\\lib',
 'C:\\Python25\\lib\\plat-win',
 'C:\\Python25\\lib\\lib-tk',
 'C:\\Python25',
 'C:\\Python25\\lib\\site-packages']
```

Chemins configurés par défaut dans Blender 2.46 :

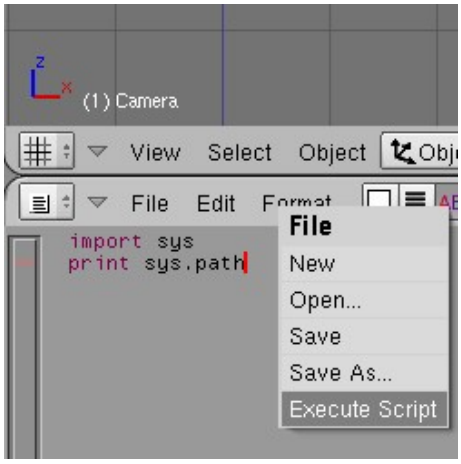


FIG. 4.101 – Récupération de la valeur de la variable sys.path avec Blender

On récupère la liste des chemins suivants dans la console :

```
"['C:\\Python25\\Lib',
  'C:\\Python25\\DLLs',
  'C:\\Python25\\Lib\\lib-tk',
  'C:\\Python25',
  'C:\\Python25\\lib\\site-packages',
  'C:\\_Stage_VV\\Code\\scripts_python_Blender',
  'C:\\PROGRA~1\\Blender',
  'C:\\Program Files\\Blender',
  'C:\\Program Files\\Blender\\python25.zip',
  'C:\\Program Files\\Blender\\.blender\\scripts',
  'C:\\Program Files\\Blender\\.blender\\scripts\\bpymodules']"
```

Mes programmes SOFA pour Blender sont donc exclusivement dans le dossier :

`C:\\_Stage_VV\\Code\\scripts_python_Blender\`

Pour appeler des programmes qui sont dans des sous-dossiers de “scripts_python_Blender”, j’ajoute le dossier voulu dans le chemin de sys.path et je le retire en fin de programme pour ne pas polluer les autres scripts qui utiliseraient des modules de même nom. Ce qui donne en Python :

Listing 4.11 – Modification et rétablissement du chemin

```
# Nom du dossier des scripts utilisateurs
```

```
2 strNomDossierScriptsUser = Blender.Get('uscriptsdir')
  # Modification du sys.path
4 strNomDossierInclude = os.path.join( strNomDossierScriptsUser , \
                         "Import_de_SOFA_v0_01" )# <<< CONFIGURER ICI <<<<
6 if not strNomDossierInclude in sys.path:
    sys.path.append( strNomDossierInclude )
8
  # Appel des fonctions de "Import_de_SOFA_v0_01"
10 import main_CImport_depuis_SOFA
  reload( main_CImport_depuis_SOFA )
12 from main_CImport_depuis_SOFA import *

14 # Rétablissement du chemin original pour ne pas "polluer" d'autres fonctions
  while strNomDossierInclude in sys.path:
16   sys.path.remove( strNomDossierInclude )
```

Utilisation de la documentation de l'API Blender pour Python

A la ligne 2 du listing 4.11 ci-dessus, on note l'instruction :

`Blender.Get('uscriptsdir')`

Pour trouver la documentation sur l'interface Blender il faut lire la documentation Blender à cet endroit :

http://www.blender.org/documentation/246PythonDoc/

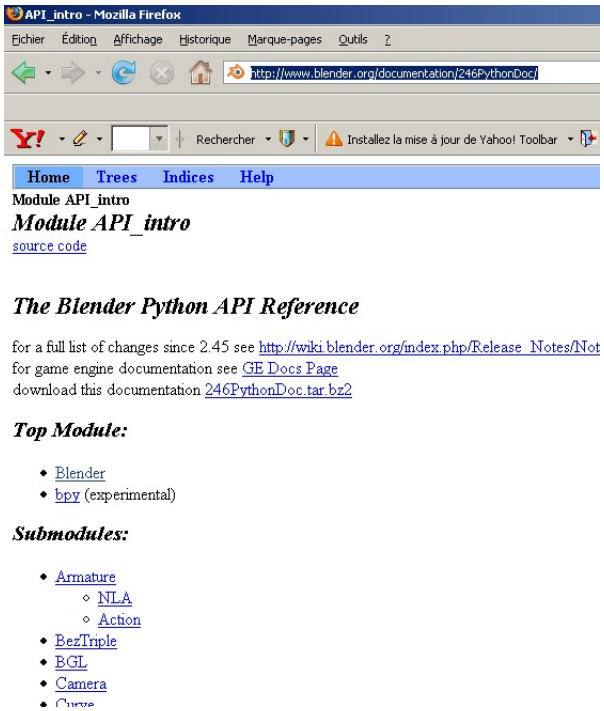
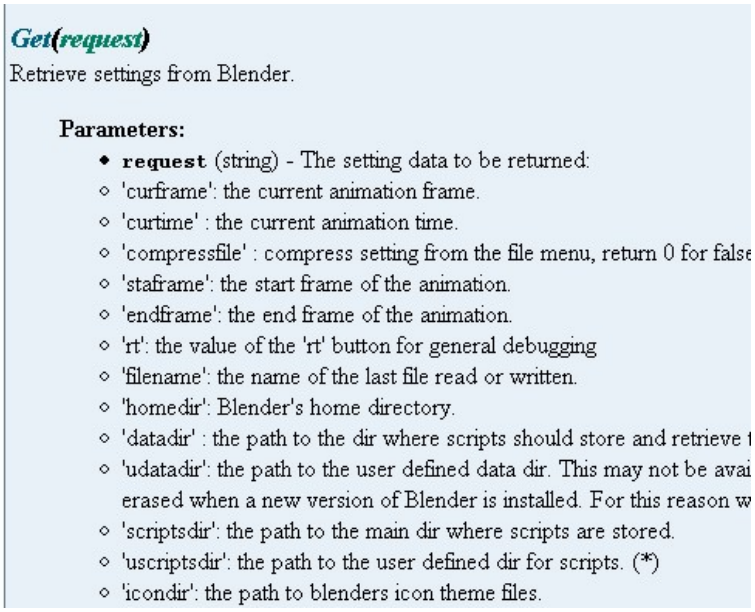


FIG. 4.102 – Page d'accueil de la documentation de l'API Blender

Ensuite, si on clique sur "Blender", puis "Get", on trouve cette page :



Get(request)

Retrieve settings from Blender.

Parameters:

- **request** (string) - The setting data to be returned:
  - 'curframe': the current animation frame.
  - 'curtime' : the current animation time.
  - 'compressfile' : compress setting from the file menu, return 0 for false
  - 'staframe': the start frame of the animation.
  - 'endframe': the end frame of the animation.
  - 'rt': the value of the 'rt' button for general debugging
  - 'filename': the name of the last file read or written.
  - 'homedir': Blender's home directory.
  - 'datadir' : the path to the dir where scripts should store and retrieve
  - 'udatadir': the path to the user defined data dir. This may not be avai erased when a new version of Blender is installed. For this reason w
  - 'scriptsdir': the path to the main dir where scripts are stored.
  - 'uscriptsdir': the path to the user defined dir for scripts. (*)
  - 'icondir': the path to blenders icon theme files.

FIG. 4.103 – Détail de la fonction “Get” du module Blender

Avec les scripts livrés par défaut avec Blender, on trouve de nombreux exemples qu’il est facile de récupérer et adapter à ses besoins.

4.14 Ajout de courbes de Bézier dans une scène Blender

Cet exemple montre comment ajouter des ligaments (représentés par une courbe de Bézier) dans une scène Blender, avec un script Python.

Voici un exemple de code Python pour Blender et le résultat correspondant.

Listing 4.12 – Ajout d'une courbe de Bézier dans une scène Blender

```
import Blender
from Blender import Object, Curve, BezTriple, Scene

[...]

  def ajouter_Bezier( self ):
    """Ajoute une courbe de Bézier dans la scène courante"""
    curve = Curve.New()
    curve.setResolu(128)# résolution dans Blender (12 par défaut)
    bt1 = BezTriple.New( -0.2, -0.1, 0.0,   -0.1, 0.0, 0.0,   0.0,  0.1, 0.0 )
    bt2 = BezTriple.New(  0.1,  0.1, 0.0,    0.2, 0.0, 0.0,   0.3, -0.1, 0.0 )
    curve.appendNurb( bt1 )
    curve[0].append( bt2 )
    scene = Scene.GetCurrent()
    ob = scene.objects.new( curve )
    ob.setLocation( 0.2, 0.0, 0.0 )
    ob.setName( "ma_courbe" )
```

Voici, figures 4.104 et 4.105, le résultat correspondant au listing 4.12 ci-dessus (la courbe est dans le plan XY et on voit l'origine de ces axes à gauche des figures).

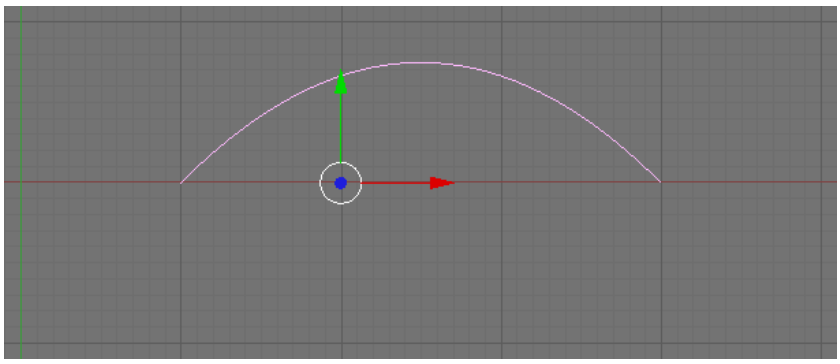


FIG. 4.104 – Position de la courbe de Bézier (en $0.2, 0.0, 0.0$)

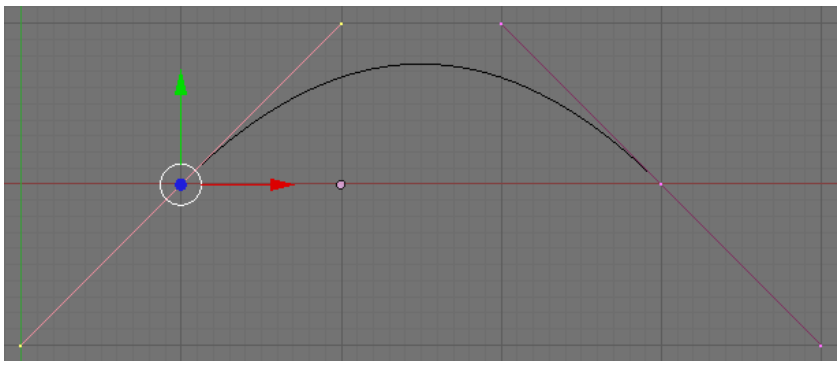


FIG. 4.105 – Courbe de Bézier en mode édition (appui sur la touche [TAB])

4.15 Problèmes rencontrés

Voici, ci-dessous la liste des problèmes rencontrés lors de l'utilisation de SOFA en C++. Cette liste servira peut-être à dépanner d'autres stagiaires ou commencer un début de F.A.Q SOFA ?

4.15.1 Avec SOFA

L'animation "éclate" au premier pas

Le problème venait d'un quaternion initialisé à `( 0, 0, 0, 0)` au lieu de `( 0, 0, 0, 1)`

Les collisions entre deux solides rigides ne fonctionnent pas

Il faut ajouter un objet "NewProximityIntersection" dans la racine de la scène (l'objet "ProximityIntersection" est inopérant dans ce cas.

Problèmes de mapping

La ligne

```
RigidRigidMapping<
            MechanicalMapping< MechanicalState<RigidTypes>,
                               MechanicalState<Vec3Types> >
          > * mapping_attaches_tibia;
```

provoque une erreur : il ne faut pas utiliser `RigidRigidMapping`, mais `RigidMapping`

La ligne

```
RigidMapping<
            MechanicalMapping< MechanicalState<RigidTypes>,
                               MechanicalState<RigidTypes> >
          > * mapping_attaches_tibia;
```

provoque une erreur : il ne faut pas utiliser `RigidMapping`, mais `RigidRigidMapping`

L'interaction avec la touche SHIFT et la souris ne fonctionne pas

Ce problème m'est arrivé avec une scène composée de deux solides rigides, avec un solveur unique dans la racine de la scène. Le problème a été corrigé lors de l'ajout d'un solveur dans chaque noeud de l'animation. Par exemple, si le graphe contient deux nœuds avec chacun un "`MechanicalObject<RigidTypes>`", il faut ajouter un solveur dans chacun de ces nœuds.

L'animation s'arrête sur erreur interne non récupérable...

La scène s'arrête sur une erreur interne non récupérable dès l'appui sur le bouton "Animate" si on décommente la ligne 11 du listing 4.13.

Listing 4.13 – listing erreur mapping

```
  // tibia
  RigidMapping<
            MechanicalMapping< MechanicalState<RigidTypes>,
                               MechanicalState<Vec3Types> >
```

```
        > * mapping_attaches_tibia;
6  mapping_attaches_tibia = new RigidMapping<
         MechanicalMapping< MechanicalState<RigidTypes>,
8                           MechanicalState<Vec3Types> >
        >( dof_tibia, attaches_tibia );
10 mapping_attaches_tibia->setName( "mapping_attaches_tibia" );
   //node_attaches_tibia->addObject( mapping_attaches_tibia );
```

L'erreur provenait du fait qu'il y avait un solveur dans la branche tibia. Avec un solveur à la racine, l'animation se comporte correctement, sans erreurs.

La gravité ne change pas quand on la modifie dans un noeud de la scène

Effectivement, la gravité ne doit être changée que dans la racine de la scène. Elle est ensuite propagée à tous les autres noeuds enfants.

Si on la modifie dans un noeud enfant, la modification est écrasée par celle de la racine.

Pour la modifier dans un fichier XML de scène SOFA, on écrit par exemple :
`<Node name="root" gravity="0 0 -9.81" >`

Comment changer la position et l'orientation des objets dans la scène XML ?

Il faut utiliser les attributs `dx`, `dy`, `dz` et `rx`, `ry`, `rz` des objets ...

- `<object type="MechanicalObject" [...]>`,
- `<object type="MechanicalObject" [...]>`,
- `<object type="OglModel" [...]>`

... de la racine des "pièces".

Comment modifier la couleur des objets autrement que par "Color=nom de la couleur" dans la scène XML ?

Il faut écrire par exemple `<Object type="OglModel" [...] color="1.0 0 0" />` pour avoir du rouge (on donne les composantes R, G, B de zéro à un).

Sous Windows, après SVN update, la génération de la solution produit des erreurs pour certains projets

Avant d'ouvrir le fichier "...\Sofa-dev\trunk\Sofa.sln" avec Visual Studio, il
faut executer le fichier "...\Sofa-dev\trunk\Project VC8.bat".

Ensuite, à la génération du projet, il faut être en mode "Release" et pas en mode "Debug". Le mode "Debug" génère un fichier "runSOFAd.exe" qui ne s'exécute pas.

Quaternions dans SOFA

Dans SOFA, la structure des 4 scalaires utilisée pour les quaternions a subi une permutation circulaire par rapport à la littérature sur les quaternions.

4.15.2 Réglage des paramètres de la scène “Genou”

Pendant ce stage j’ai passé beaucoup de temps à faire de nombreux essais sur les différents paramètres de la scène (longueur, raideur, position des ligaments, géométrie des os, distance et raideur de contact).

J’ai aussi réussi à perdre des réglages qui fonctionnaient assez bien en voulant les modifier un peu... -sic-

4.15.3 Compatibilité des programmes Python et Blender sur les différentes machines

Compatibilité Windows-Windows

Voici les écrans d’erreurs (4.106 et 4.107) obtenus avec les scripts d’import-export des données de “myCorporisFabrica” sur ma machine personnelle (Windows 2000 pro). Alors que le même code fonctionne correctement sur le poste de l’Inria (Windows XP)... Cette erreur irrécupérable survient à la suite d’un simple produit matrice 4x4, vecteur...

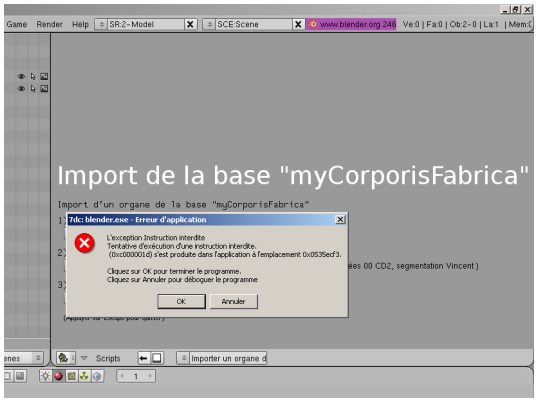


FIG. 4.106 – Erreur bloquante lors de l’import de myCorporisFabrica

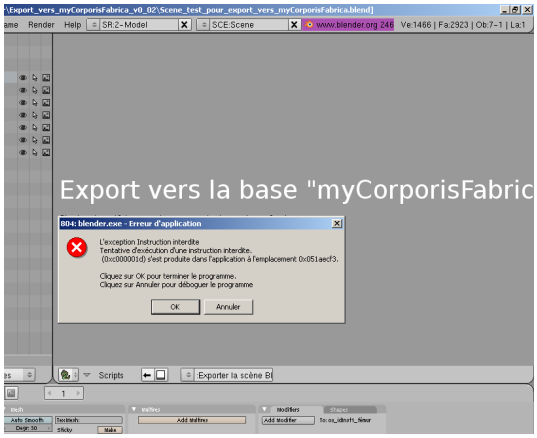


FIG. 4.107 – Erreur bloquante lors de l’export de myCorporisFabrica

Compatibilité Windows-Linux

Il reste aussi des problèmes sur les machines Kubuntu v8.04 sur lesquelles le script suivant renvoie `None` au lieu du chemin configuré pour les scripts utilisateurs :

```
1 import Blender
print Blender.Get('uscriptsdir') # Récupération du chemin des scripts utilisateurs
```

Ce problème n’est pas présent sur les machines équipées de Ubuntu 8.04.

Bibliographie

[1] Physiologie Articulaire, membre inférieur", A.I. Kapandji, éditions Maloine (5^{e} édition), 2007, ISBN (10) 2-224-01052-4

[2] Anatomie du membre inférieur, Chapitre 15 : Ensemble fonctionnel du genou, Docteur Olivier Palombi, Année universitaire 2007/2008, Faculté de Médecine de Grenoble (UJF)

[3] Contribution à la modélisation de l'articulation du genou : outils géométriques et cinématiques, (Thèse présentée et soutenue publiquement le 7 décembre 2007 pour l'obtention du Doctorat de l'Université de Reims Champagne-Ardenne (URCA) (Spécialité Informatique)) par Antoine Jonquet