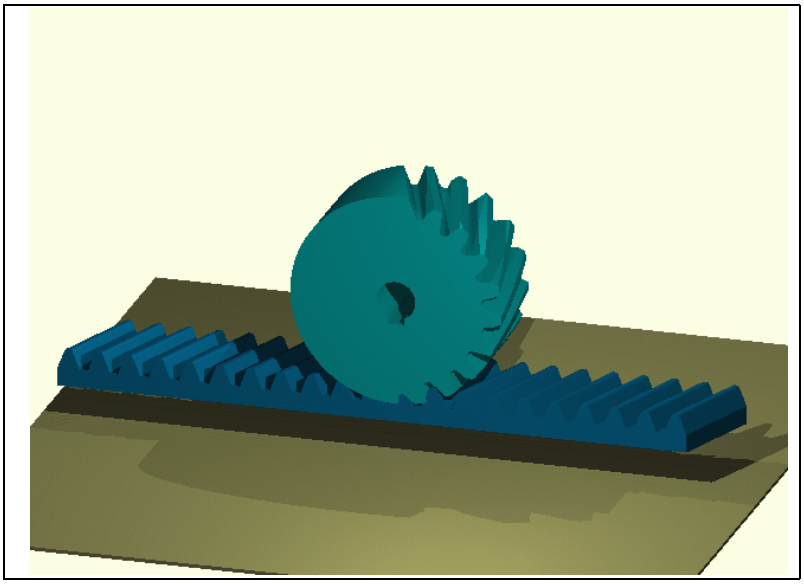
Année 1998-99

Accès libre

ADAPTATION D'UN LOGICIEL DE MODÉLISATION VOLUMIQUE À LA REPRÉSENTATION D'ENGRENAGES



Auteurs : M. ARMENJON

M. GUENIFFEY

Directeurs de PFE : M. BONIAU (Cluny)

M. VAUDELIN (Paris)

GRCAO

SGDL

Notice bibliographique

ANNEE : 1999 **GROUPE :** EEA **NUMERO DU PFE :** 3301

CENTRE DE RATTACHEMENT DU PFE : ENSAM Cluny

AUTEURS : M. ARMENJON Frédéric & M. GUENIFFEY David

TITRE : Adaptation d'un logiciel de modélisation volumique à la représentation d'engrenages

ENCADREMENT DU PFE : M. BONIAU (ENSAM Cluny) & M. VAUDELIN (ENSAM Paris)

PARTENAIRES DU PFE :
– M. ROTGÉ, SGDL Systèmes Inc.
– M. TIDAFI, GRCAO (Université de Montréal)
– M. OCTRUE, CETIM

NOMBRE DE PAGES : 54 **NOMBRE DE REFERENCES BIBLIOGRAPHIQUES :** 18

RESUME : Le projet est à caractère de veille technologique pour le CETIM : il a pour but de démontrer l'intérêt de l'utilisation du logiciel SGDLsoft, un logiciel de modélisation volumique fondé sur la géométrie projective et toujours en développement à Montréal à cette date, en termes de représentation informatique d'engrenages. Ce logiciel de modélisation volumique est fondé sur la géométrie projective et continue d'être en développement à Montréal à cette date. Le projet traite plus particulièrement des engrenages hélicoïdaux en développante de cercle.

MOTS-CLES : CAO, CREMAILLERE, ENGRENAGES, ENGRENAGES HELICOIDAUX, GEOMETRIE PROJECTIVE, MODELISATION VOLUMIQUE, QUADRIQUES, TAILLAGE

PARTIE A REMPLIR PAR LE PROFESSEUR RESPONSABLE DU PROJET		
ACCESSIBILITE DE CE RAPPORT (entourer la mention correcte) :		
LIBRE	CONFIDENTIEL pendant	an(s)
DATE :	Nom du signataire :	Signature :

Table des matières

Table des figures

Remerciements

Nous aimerions remercier avant tout messieurs Jean-François ROTGÉ et Henri BONIAU, sans qui notre projet n'aurait jamais pu voir le jour. Nous tenons à exprimer en particulier notre reconnaissance à M. ROTGÉ pour nous avoir permis de partager son enthousiasme, sa vision originale de la géométrie et de l'espace, mais aussi pour avoir su jalonner notre travail de ses conseils, suggestions et mises en garde.

De plus, il nous aurait été impossible de pousser aussi loin notre étude sans l'aide et le soutien technique de messieurs Laurent DANIEL et Olivier BELLIS, de la société S.G.D.L. Systèmes Inc., qui ont toujours su rester disponibles quand nous en avions besoin.

Par ailleurs, nous tenons à associer à ces remerciements l'ensemble des membres du GRCAO (Groupe de Recherche en Conception Assistée par Ordinateur) de l'Université de Montréal ; merci tout particulièrement à monsieur Temy TIDAFI pour avoir bien voulu nous accueillir au sein de son "équipe" et nous avoir donné les moyens de réaliser notre travail dans les meilleures conditions possibles, merci aussi à monsieur Claude PARISEL pour l'intérêt qu'il a manifesté pour notre projet et les discussions qui en ont découlé ; merci aussi aux autres membres (Ivanka IORDANOVA, Denis GAMACHE, Marius BOGDAN…) pour leur accueil chaleureux. Une dernière attention pour monsieur Roger CAMOUS, qui s'est montré compréhensif envers nous lors de nos démarches administratives.

Enfin, il ne nous reste plus qu'à remercier monsieur Michel OCTRUE, notre interlocuteur au CETIM (Centre Technique des Industries Mécaniques), pour sa contribution dans l'élaboration de notre sujet et de nos principaux objectifs.

Encore une fois, merci à tous…

Notations

En ce qui concerne les définitions des éléments géométriques et les symboles, nous avons essayé d'utiliser au maximum les normes ISO relatives aux engrenages (ISO 1122 et 701) [6, p. 55-58].

Principaux symboles géométriques

a	Entraxe
b	Largeur de denture
d	Diamètre
g	Longueur (de conduite, de recouvrement…)
h	Hauteur de dent, saillie, creux
m	Module
p	Pas
r	Rayon
s	Epaisseur de dent
x	Coefficient de déport
Z	Nombre de dents
α	Angle de pression
β	Angle d'hélice
ϵ	Rapport (de conduite, de recouvrement…)
inv α	Involute α ($\tan\alpha - \alpha$)

Indices additionnels

a	de tête
b	de base, d'approche
f	de pied, de retraite
n	réel (c-à-d dans une section droite)
t	apparent
α	de conduite
β	de recouvrement, d'hélice
0	relatif à l'outil
1	relatif au pignon
2	relatif à la roue

$()'$	de fonctionnement (c-à-d avec déport)
$()$	de référence, de taillage
inv	relatif à la développante

Principaux symboles composés

a	Entraxe normal, ou de référence
a'	Entraxe de fonctionnement
d	Diamètre primitif de denture, de référence ou de taillage
d'	Diamètre primitif de fonctionnement
d_a	Diamètre de tête
d_f	Diamètre de pied
d_b	Diamètre de base
g_f	Longueur d'approche
g_a	Longueur de retraite
g_α	Longueur d'action
g_β	Longueur de recouvrement
h	Hauteur de dent
h_a	Saillie
h_f	Creux
m_0	Module d'outil
m'	Module de fonctionnment
m_n	Module réel
m_t	Module apparent
m_{t0}	Module d'outil apparent
m'_t	Module apparent de fonctionnement
p	Pas (primitif)
p_0	Pas d'outil
p_b	Pas de base
p_n	Pas réel (primitif)
p_t	Pas apparent(primitif)
r_{inv}	Rayon du point limite de la développante
r_A	Rayon actif de pied
s_a	Epaisseur de tête
s_f	Epaisseur de pied
s_b	Epaisseur de base
s_n, s_t	Epaisseur réelle, apparente
x_1	Déport du pignon
x_2	Déport de la roue
Z_1 , Z_2	Nombre de dents
α_0	Angle de pression d'outil

α'	Angle de pression de fonctionnement
α_n	Angle de pression réel
α_t	Angle de pression apparent
α_{t0}	Angle de pression apparent d'outil
α'_t	Angle de pression apparent de fonctionnement
β_0	Angle d'inclinaison d'outil
β'	Angle d'inclinaison de fonctionnement
ϵ_α	Rapport de conduite
ϵ_β	Rapport de recouvrement

La crémaillère de taillage

Les éléments de la crémaillère génératrice normalisée, dans le cas d'une denture hélicoïdale, sont les éléments réels : le module m_0 est donc le *module réel d'outil*, et l'angle de pression α_0 = 20° , *l'angle de pression réel d'outil.*

Nous avons aussi utilisé deux symboles ne figurant pas dans la norme, mais dans un ouvrage de M. VEDMAR [17, p. 7] : ce sont deux facteurs multiplicatifs, r_{0t} et h_{0t}, caractérisant le rayon en tête de dent pour r_{0t} ($r = r_{0t} \cdot m_0$) et la hauteur de la dent pour h_{0t}.

Enfin, pour se ramener au cas le plus classique, nous allons considérer tout au long de notre étude que $r_{0t} = 0{,}35$ et $h_{0t} = 1{,}25$, amenant ainsi la hauteur de référence de la dent à la grandeur bien connue $2{,}25 \cdot m_0$.

Première partie

Développement

Chapitre 1

Introduction

1.1 Problématique et objectifs

Contrairement à ce que l'on pourrait croire, la révolution informatique des vingt dernières années n'a pas conduit, de manière générale, à la révolution des méthodes de travail à laquelle on aurait pu s'attendre, on a en effet plutôt réalisé un transfert d'un outil à un autre de ces méthodes, sans en remettre en cause les fondements : il y a eu une translation des capacités, et non une réelle réflexion sur les potentialités et les exigences de l'outil informatique, ce qui s'est traduit dans le domaine des engrenages par une informatisation des abaques par exemple, sans véritablement ré-envisager l'engrenage par rapport son mode de représentation.

Ainsi, actuellement, comment représente-t-on un engrenage sur un logiciel de CAO ? De manière simplifiée, on commence par représenter en 2D une section, à laquelle on demande de suivre une trajectoire définie dans la 3^{e} dimension en fonction de l'engrenage voulu (droit, hélicoïdal ...) selon un pas donné et l'ordinateur détermine alors la surface passant par ces différentes sections (NURBS ou autres), sans que l'utilisateur n'ait un véritable contrôle de cette surface.

Cette modélisation est intéressante, mais elle est aussi très lourde d'un point de vue arithmétique et donc en termes de calculs et ressources machine, mais surtout elle n'intègre pas intrinsèquement la notion de volume. Aussi, disposer d'un modeleur véritablement volumique dans un environnement de programmation puissant (c'est le cas du logiciel SGDLsoft) nous semblait propice à développer une approche résolument novatrice en matière de représentation d'engrenages.

Dans cette optique, nous avons contacté le CETIM (Centre Technique des Industries Mécaniques), afin de disposer du soutien industriel de spécialistes en la matière. Notre contact là-bas, M. OCTRUE (directeur technique et spécialiste des roues et vis sans fin), a alors envisagé avec nous une action de veille technologique pour déterminer l'intérêt d'une telle démarche. De notre entretien avec lui a découlé notre sujet et nos principaux objectifs :

- tenter de représenter deux engrenages hélicoïdaux à axes perpendiculaires, avec leurs dents à profil en développante de cercle, en "simulant" leur mode de génération ;
- valider cette représentation en générant le même engrenage à l'aide d'un modèle mathématique et tenter de retrouver la ligne d'engrènement.

La validation de tels objectifs pourrait alors permettre d'ouvrir de nouvelles perspectives en matière de représentation des engrenages (étude plus approfondie du contact...), d'où l'intérêt de notre projet.

1.2 SGDLsoft par rapport à l'approche traditionnelle en CAO.

Aujourd'hui, les systèmes de représentation 3D disposent tous de techniques de rendu réaliste permettant la réalisation d'images de synthèse de scènes complexes, bien que toujours conditionnées par la nature mathématique et informatique des informations géométriques stockées. Et pour générer des images extrêmement réalistes, la modélisation polygonale reste la plus adaptée, même si elle est l'une des représentations les plus simples de notre environnement géométrique.

Toutefois, la description géométrique approximative des objets à partir de polygones limite les applications de cette technique à la simulation visuelle (effets spéciaux, infographie...). Et la notion d'objets 3D mathématiquement précis et bien définis devient fondamentale pour des applications industrielles et scientifiques. La description polygonale ne suffit alors plus, elle est abandonnée au profit d'une description surfacique des objets. Par exemple, une sphère approximée par un ensemble de polygones sera décrite par son équation mathématique, tous les calculs secondaires, comme le calcul du volume, auront alors une réelle fiabilité. Dans ce cas, la description mathématique et les calculs géométriques permettent de simuler la notion d'intérieur des objets, et par conséquent de les sectionner. Il s'agit d'un modèle surfacique solide dit B-Rep (représentation par les bords).

Parvenu à ce stade de modélisation, tout utilisateur sera confronté à une série de problèmes techniques inhérents à la technologie B-Rep, allant de la "boulimie" de mémoire destinée au stockage informatique de l'information, à l'absence de la reconnaissance implicite des notions de plein ou de vide, et donc de matière, en passant par l'impossibilité de localiser exactement ce qui est à l'intérieur, à l'extérieur ou à la surface de l'objet.

C'est à ce stade de modélisation qu'intervient le logiciel SGDLsoft. En effet, se basant sur des méthodes mathématiques et informatiques tout à fait novatrices et radicalement différentes des précédentes, ses concepteurs (M. ROTGÉ pour l'algorithmique géométrique et infographique et M. DANIEL pour l'architecture et le génie logiciel) ont conçu un véritable langage volumique, la technologie S.G.D.L. (pour *Solid Geometry Design Logic*). Ce qui fait de SGDLsoft à la fois :

- **un outil de programmation géométrique**, unifiant l'approche originale de ses concepteurs au langage de programmation fonctionnelle Scheme, dérivé de Lisp ;
- **un outil de modélisation volumique**, considérant simultanément la portion d'espace occupée par un objet et la forme géométrique qui le délimite, dans une configuration géométrique indépendante de ses mensurations ; l'affectation de densités de manière intrinsèque à cet objet définit la notion fondamentale de matière, là où la technologie B-Rep ne faisait que la simuler.

De telles caractéristiques font de SGDLsoft un vériatble moteur et processeur volumique projectif capable de décrire, générer, contrôler et visualiser toute scène volumique.

1.3 Limites de notre étude

Le cadre de notre étude est la validation ou non du logiciel SGDLsoft comme outil de représentation des engrenages. Or, pour pouvoir générer ces engrenages sur ordinateur, nous sommes tributaires de modèles mathématiques et physiques. Toutefois, ces modèles ont un champ d'application bien défini et leurs limitations ne sauraient être imputables au logiciel utilisé.

Ainsi, aux vues des difficultés à trouver des modèles mathématiques 3D d'une denture hélicoïdale en développante ainsi que de la complexité de ces modèles, et étant donné les délais assez serrés qui nous sont impartis, notre principal objectif n'est pas la remise en cause de tel ou tel modèle, ni la détermination d'un nouveau modèle, mais il consiste plutôt en la validation du potentiel de SGDLsoft en matière de représentation d'engrenages, moyennant la connaissance de modélisations particulières que l'on puisse transcrire sous le langage de programmation volumique du-dit logiciel.

Toutefois, une attention particulière a été portée au choix des modèles, à la fois pour garantir des résultats les plus proches possibles de la réalité (prise en compte de l'influence de la géométrie de l'outil de taillage sur le profil de l'engrenage, possibilité de sortie des principales caractéristiques des engrenages comme le rapport de conduite, la longueur de recouvrement…), tout en gardant la plus grande cohérence possible au niveau des notations (on a en particulier essayé d'utiliser au maximum les notations ISO).

Chapitre 2

La méthodologie employée

2.1 Le planning et le suivi du projet

Dès le début de notre projet, nous avons voulu nous placer dans une démarche structurée en termes d'objectifs et de réactivité vis-à-vis de l'évolution de notre travail. En effet, comme nous l'avons vu en 1.1, notre projet s'inscrit dans une perspective de veille technologique pour le CETIM, autrement dit notre étude porte sur des éléments dont nous n'avions *a priori* aucune certitude en termes de résultats.

Malgré les difficultés pour tenir un planning dans un contexte incertain comme celui-ci, nous avons tout de même tenu à nous en fixer un en termes d'objectifs. Finalement, notre projet a eu pour planning celui illustré en figure 2.1. Nous avons réussi à tenir notre planning pour tout ce qui est structuration de notre travail (réunions et rapports quotidiens), mais pour le reste, les prévisions étaient vraiment impossibles.

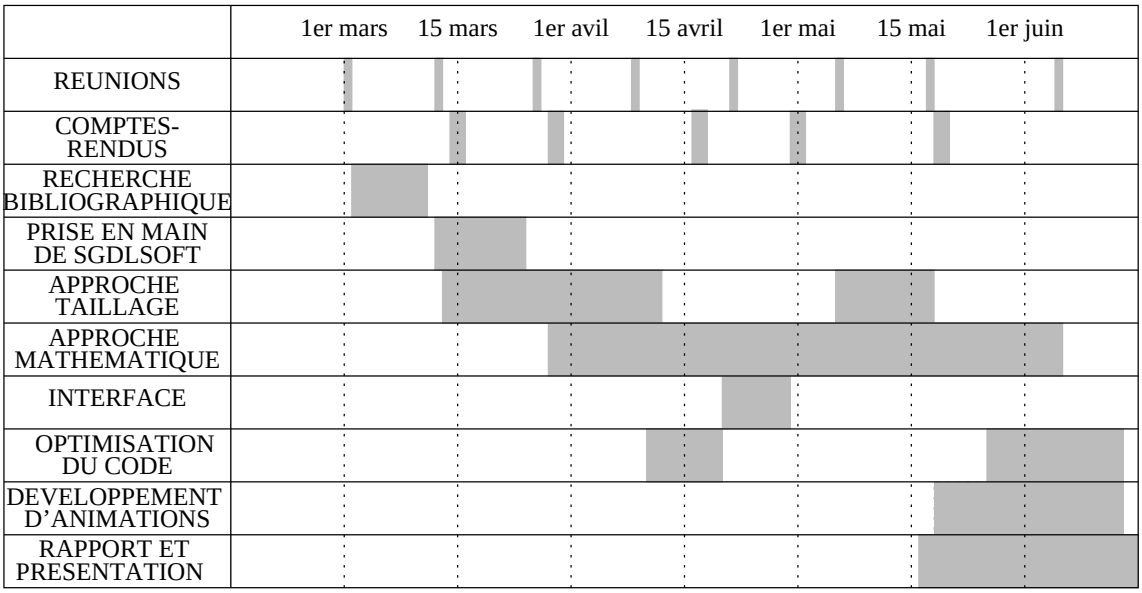


FIG. 2.1 – *Planning de notre projet*

C'est aussi dans l'optique d'un contexte incertain quant aux résultats qu'il était important pour nous de pouvoir être réactif vis-à-vis de nos orientations au cas où nous déboucherions sur une impasse. C'est la raison pour laquelle nous avons décidé d'adopter une politique de suivi du projet cohérente, afin que l'information circule dans les meilleures conditions entre les différents acteurs de notre projet. Cette décision s'est traduit en particulier par la mise en place avec M. ROTGÉ de réunions hebdomadaires, où nous pouvions faire le point et nous recentrer sur l'essentiel, ainsi que par des comptes-rendus réguliers (une fois toutes les 2 semaines) avec nos différents correspondants en France : notre commanditaire, M. OCTRUE, ainsi que nos responsables de projet, M. BONIAU à Cluny et M. VAUDELIN à Paris.

2.2 La recherche bibliographique

A notre arrivée à Montréal, malgré la période de préparation du premier semestre, plutôt mise à profit pour cadrer correctement notre sujet et régler les différents problèmes administratifs, nous manquions tout de même relativement de recul vis-à-vis des engrenages hélicoïdaux et leur problématique. Dans le cadre de notre première réunion avec M. ROTGÉ, nous avons donc décidé de passer la première semaine de notre séjour à réaliser une recherche bibliographique poussée en la matière.

Et la richesse des bibliothèques des différentes universités montréalaises (Université de Montréal, Ecole Polytechnique de Montréal, Ecole de Technologie Supérieure, Université Mac Gill, Université Concordia) est suffisamment impressionnante pour que notre recherche n'ait pas été une perte de temps : en effet, cette recherche nous a tout d'abord permis de bien nous mettre à jour sur les engrenages et les problématiques qui y sont liées, elle nous a aussi permis d'explorer en partie la documentation américaine en la matière, documentation à laquelle il n'est pas facile d'avoir accès en France, et elle nous a enfin permis d'éviter de refaire des études qui auraient déjà été traitées.

Finalement, nous avons retrouvé un certain nombre de références que nous avions déjà repérées en France, mais un grand nombre des autres références concernait la littérature anglophone, que nous n'avions pas eu l'opportunité d'étudier auparavant. Parmi ces références, celles qui nous ont servi au sens strict du terme sont reportées dans la bibliographie de notre rapport, mais un certain nombre, si elles ne nous ont pas servi directement, nous ont quand même permis de revoir les fondements de notre culture technologique en matière d'engrenages ; c'est le cas en particulier des ouvrages suivants :

- *Manual of gear design*, Earle BUCKINGHAM, New-York Industrial Press 1935-37 ;
- *Analytical mechanics of gears*, Earle BUCKINGHAM, 1re édition, New-York McGraw-Hill 1949 ;
- *Gear handbook : the design, manufacture and application of gears*, Darle W. DUDLEY, New-york McGraw-Hill 1962 ;
- *Le formulaire des engrenages : traité pratique pour le calcul, le tracé et l'exécution des engrenages*, Charles MACABRAY, 5e édition, Paris Dunod 1958.

Nous avons aussi pu avoir accès à un certain nombre d'articles et mémos qui nous ont permis de nous faire une idée des derniers développements en matière d'engrenages hélicoïdaux. Voici les

principaux (mis à part ceux de la bibliographie du rapport bien entendu) :

- *A CAD approach to helical groove machining : mathematical model and model solution*, K.F. EHMANN, S.K. KANG & C. LIN, International Journal Machine Tools & Manufacture, Jan. 1996, 36 (1), p.141-153 ;
- *Profiling of rotation tools for forming of helical surfaces*, G. NANKOV & V. IVANOV, International Journal Machine Tools & Manufacture, Sep. 1998, 38 (9), p.1125-1148.

En conclusion, nous n'avons trouvé que peu de documentation réellement pertinente pour la représentation tridimensionnelle informatique des engrenages hélicoïdaux, même àu niveau mondial. Les modèles à notre disposition se limite finalement aux travaux de M. LITVIN [11, 12], ainsi qu'à ceux de M. LAURIA [10]. Mais cela nous conforte dans l'intérêt de notre projet et son caractère véritablement pointu en matière de représentation d'engrenages.

2.3 L'organisation du travail

Pour notre projet, la grosse difficulté est d'arriver à représenter correctement une denture hélicoïdale en développante de cercle. Il est vrai que l'hélice comme la développante de cercle sont 2 courbes bien définies mathématiquement, mais le problème se situe au niveau de la représentation de ces courbes sur un ordinateur : ainsi, une hélice obéit à une équation implicite polynômiale de degré infini, ce qui signifie que l'ordinateur ne peut qu'en représenter une approximation, même si c'est une bonne approximation, et les logiciels qui tracent des surfaces hélicoïdales font forcément une approximation à un endroit ou à un autre.

Il faut bien garder à l'esprit cette donnée pour envisager correctement les différentes approches possibles. Celles-ci sont d'ailleurs globalement au nombre de 2 :

- soit on essaye de s'affranchir de cette hélice en optant pour une approche simulant le mode d'élaboration de l'engrenage (roulement sans glissement...) en se libérant ainsi complètement de l'aspect mathématique du problème, c'est une approche plutôt orientée enlèvement de matière, on gère le vide : cette approche est celle que nous avons abordée en premier lieu, elle est développée au chapitre 4.
- soit au contraire on préfère régler le problème de manière plus mathématique, en contrôlant toutes les approximations effectuées, grâce à l'utilisation de modèles mathématiques, dans ces conditions on a une approche plutôt orientée matière, on ne gère plus le vide mais le plein : c'est l'approche que nous avons envisagée dans un deuxième temps, elle est développée au chapitre 5.

Mais comme toujours avec l'outil informatique, il existe des problèmes qui lui sont spécifiques, indépendemment de l'approche envisagée. Ainsi, tout algorithme, tout code informatique nécessite une étape d'optimisation pour obtenir les meilleures performances de la machine. Ce travail a bien entendu été effectué en parallèle des 2 approches précédentes, mais étant donné l'étendue de ce travail nous avons jugé nécessaire de lui consacrer un chapitre entier, le chapitre 6.

De plus, une fois toute cette partie de développement du code terminée, nous avons pu commencer à réaliser quelques applications qui nous avaient été demandées par nos différents responsables

et envisager ensuite les perspectives possibles pour une suite éventuelle à notre étude. Cette partie de notre travail a été reportée au chapitre 7.

Enfin, il a fallu faire le bilan de toutes ces différentes approches, évaluer *a posteriori* les capacités du logiciel SGDLsoft et mettre celles-ci en exergue, tout en dégageant les voies encore à explorer suite à notre étude, qui n'avait finalement pour vocation que défricher un terrain somme toute relativement vierge. Cette synthèse est réalisée au chapitre 8.

Chapitre 3

Les outils utilisés

3.1 Le logiciel de modélisation volumique SGDLsoft

Plusieurs concepts-clés sous-tendent la richesse algorithmique et volumique du logiciel SGDLsoft, c'est la raison pour laquelle nous allons nous y attarder quelque peu, ce afin d'avoir les idées claires sur ce qui fondent tout notre travail par la suite.

3.1.1 La géométrie projective

D'un point de vue purement géométrique, la géométrie projective occupe une place privilégiée dans la classification des différentes géométries ; on a en effet :

$$\text{Métrique} \subset \text{Euclidienne} \subset \text{Affine} \subset \text{Projective}$$

Cette classification suggère d'un point de vue informatique la possibilité de régler algorithmiquement tous les problèmes de géométries de niveau inférieur avec la seule aide des règles projectives.

Dans ce cas, il est possible de substituer à une programmation classique essentiellement métrique, une programmation de type projectif, éliminant par la même occasion tous les cas particuliers issus des géométries inférieures.

Et pour mieux comprendre l'intérêt d'un tel type de programmation, revenir à la théorie de la construction géométrique, qui classifie la nature des problèmes géométriques à l'aide d'instruments de dessin spécifiques permettant de les résoudre, peut permettre de fixer les idées [15, § G-IV.3]:

- la géométrie métrique est essentiellement celle des distances et des angles, le compas en étant l'instrument de base ;
- la géométrie euclidienne est la géométrie des rapports d'échelle entre dessins morphologiquement identiques et le panthographe lui est associé comme instrument de base.
- la géométrie affine est quant à elle essentiellement celle du parallélisme et son instrument de base sera la règle à bords parallèles.
- enfin la géométrie projective peut se résumer à la géométrie du point, de la droite, du plan et des propriétés qui s'y rattachent, l'instrument de base étant alors la règle non graduée à un seul bord.

Ainsi, une construction projective ne nécessite qu'un crayon et une règle non graduée, ce qui fait que toute programmation reproduisant ce processus se caractérise par l'absence de notion d'infini, de parallélisme, de proportion, de distance et d'angle. Cela peut sembler une faiblesse, mais c'est en réalité le résultat d'une puissance conceptuelle visant à éliminer les cas particuliers géométriques. C'est ce principe qui régit toute construction volumique sous SGDLsoft.

3.1.2 Le système de coordonnées

Pour pouvoir bénéficier de tous les avantages de la programmation projective, il est impératif de bien maîtriser l'utilisation des coordonnées. Et le système de coordonnées utilisé à cette fin est celui de Grassmann-Plücker (on parle alors de coordonnées homogènes), dans lequel un point, une droite et un plan ont respectivement 4, 6 et 4 coordonnées.

Il peut être intéressant de faire un parallèle avec des coordonnées euclidiennes non-homogènes. Dans cette optique, on utilise une représentation du point en 3 dimensions de la forme $(X\ Y\ Z)$. L'ajout à ce triplet d'une quatrième coordonnée égale à 1 ne modifie en rien le bon fonctionnement de toutes les procédures géométriques dites euclidiennes. Les 3 premières coordonnées auront indifféremment une représentation en nombres entiers ou en nombres décimaux, pourvu que la quatrième coordonnée reste égale à 1. Cette dernière coordonnée par contre facilitera les nombreuses opérations matricielles réalisées par le logiciel, comme les projections ou les perspectives.

En particularisant la quatrième coordonnée et en lui donnant la valeur 0, on passe alors d'un système de coordonnées euclidiennes à un système de coordonnées affines, dans lequel il devient possible de contrôler et d'utiliser les points à l'infini. Ainsi, un observateur situé en (1 1 1 0) sera positionné à l'infini dans la direction déterminée par sa position euclidienne (1 1 1 1) et l'origine du repère (0 0 0 1).

Outre la possibilité de gérer l'infini de manière naturelle, cette approche permet de régler le problème de la représentation et le calcul de coordonnées rationnelles, ce qui est fondamental pour obtenir une précision parfaite dans les cas où les coordonnées sont initialement rationnelles ou amenées à le devenir. Considérons par exemple le point de coordonnées homogènes (1/3 5/6 0 1) ; il sera représenté par le quadruplet approché (0.333 0.833 0 1) ou, beaucoup mieux, par le quadruplet (2/6 5/6 0/6 6/6) $\rightarrow$ (2 : 5 : 0 : 6). Cette notion de représentation "rationnelle" des coordonnées est étendue dans le langage SGDL à la notion de distance euclidienne entre 2 points, qui devra être spécifiée par une représentation homogène : le rayon d'une sphère unité s'écrira (1 : 1) [14, § 9.2.2].

Ainsi, une distance dans le système de coordonnées de SGDLsoft est spécifiée par un doublet de coordonnées homogènes (on parle d'un V2), un point par un V4 (un quadruplet de coordonnées homogènes). Un plan sera aussi défini par un V4, par contre l'écriture directe des coordonnées de ce V4 est assez délicate. En effet, seuls les plans passant par l'origine offre une transcription directe dans un repère euclidien : d'un point de vue pratique, le plan $(a : b : c : 0)$ aura une orientation définie par $(a : b : c)$, vecteur normal, et passera par l'origine du repère euclidien. Pour obtenir un plan quelconque, il suffit de faire appel aux fonctionnalités SGDL telles que le calcul d'un plan passant par trois points ou bien encore celui d'un plan passant par un point et parallèle à un autre plan (passant par l'origine par exemple). Enfin, une droite est définie par un V6 (un sextuplet de coordonnées homogènes) : nous laisserons de côté la justification théorique de cette écriture, mais pour obtenir ce V6 il suffit de faire appel, une fois encore, aux fonctions de SGDLsoft. Par exemple,

une droite peut être définie par 2 points ; d'ailleurs dans ce cas un des points peut être à l'infini, ce qui revient à définir la droite par un point et un vecteur directeur [15, § P-IV.3].

3.1.3 Les quadriques

La forme ou primitive élémentaire actuelle est la quadrique, c'est-à-dire la surface du second degré. Généralement, une telle surface est coupée par une droite sécante en 2 points, de même que l'on peut lui mener d'une droite extérieure 2 plans tangents. La courbe plane issue de l'intersection d'un plan avec la quadrique est une section conique. L'équation générale d'une quadrique projective est la suivante :

$$a_0.x_0^2 + a_1.x_0.x_1 + a_2.x_1^2 + a_3.x_0.x_2 + a_4.x_1.x_2 + a_5.x_2^2 + a_6.x_0.x_3 + a_7.x_1.x_3 + a_8.x_2.x_3 + a_9.x_3^2 = 0$$

L'initialisation des 10 coefficients, $a_0, \ldots, a_9$, permet de définir numériquement une quadrique quelconque de l'espace projectif tridimensionnel et par conséquent d'en obtenir la représentation informatique à l'écran, à l'aide d'un logiciel doté d'un moteur géométrique projectif. C'est le cas de SGDLsoft qui permet de bénéficier de la puissance de l'équation ci-dessus, qui englobe toutes les quadriques projectives, affines et métriques sans aucun cas particulier géométrique ou numérique (pas de division) [15, § G-II.2].

Pour faciliter l'utilisation de cette équation, SGDLsoft est muni d'une interface géométrique permettant le contrôle visuel, graphique et synthétique de la quadrique et automatisant l'initialisation des coefficients de la quadrique en géométrie projective, affine ou euclidienne. En fait, le contrôle géométrique de toute quadrique passe par l'extension en 3D du principe de construction des coniques définies par un triangle de construction P0, P1, P2 et un point de passage P4 (ce qui découle des travaux de Pascal et de son théorème qui énonce que les 3 paires de côtés opposés d'un hexagone inscrit dans une conique se coupent en 3 points alignés [15, § G-II.1]), comme le montre la figure 3.1. la quadrique sera donc construite et contrôlée à l'aide d'un "tétraèdre de construction" et de 2 points de passage situés dans les plans des 2 faces particulières du tétraèdre. Un ensemble de 6 points permettra donc de définir la quadrique à partir de 2 sections coniques s'interséquant dans l'espace aux 2 points de contact des plans tangents.

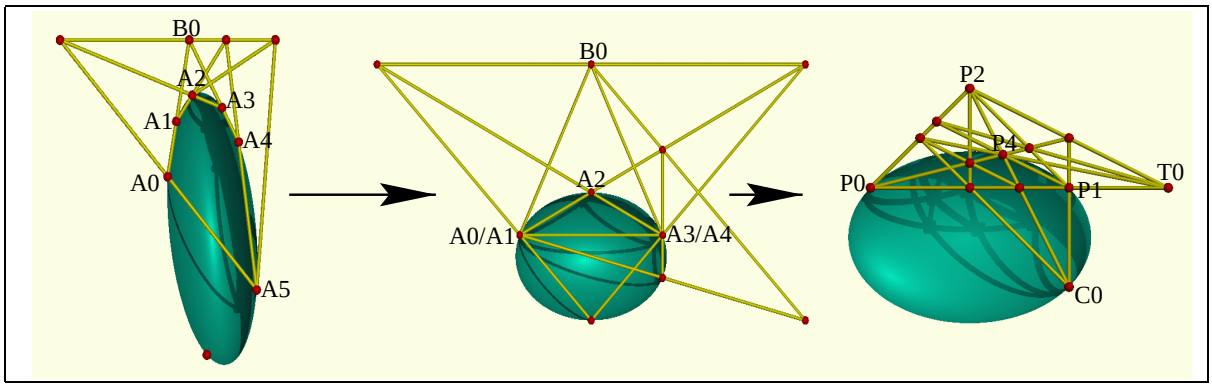


FIG. 3.1 – *Le triangle de construction P0, P1, P2 d'une conique*

Ainsi, en initialisant les 6 points de la quadrique comme l'illustre la figure 3.2, on construit la quadrique sphère de rayon Rx. Cette sphère est construite à l'aide de 2 sections circulaires appartenant à 2 plans perpendiculaires. Les points P2 et P3, qui font office de pôles, ont été rejetés à l'infini dans la direction de l'axe Y et de l'axe Z.

Donc, de manière générale, pour toute quadrique projective, affine ou métrique, on définira seulement 6 points de contrôle P0 à P5, tels que P0 et P1 soient 2 points de contacts de 2 plans tangents à la quadrique, P2 et P3 soient 2 pôles de la quadrique et en conséquence appartiennent aux 2 plans tangents et P4 et P5 soient 2 points de passage de la quadrique définissant 2 sections coniques de la quadrique dans les plans P0-P1-P2 et P0-P1-P3.

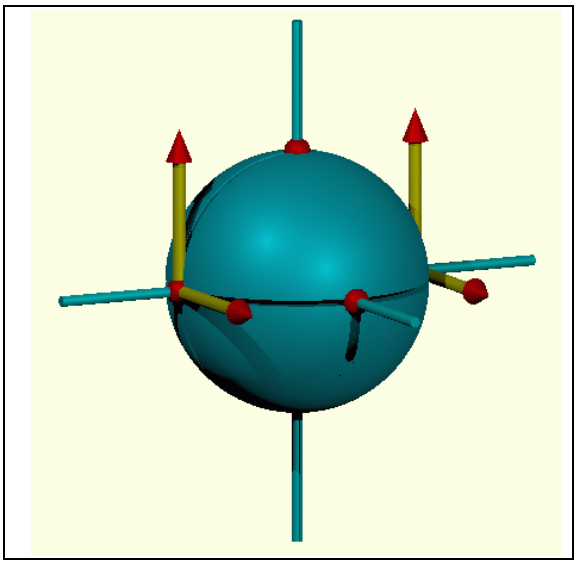


FIG. 3.2 – *Les 6 points de contrôle d'une quadrique et le passage d'une sphère à un ellipsoïde*

On peut ainsi obtenir un ellipsoïde en modifiant la position de P5 sur le demi-axe des Z, la quadrique sera alors modélisée comme sur la figure 3.2. Pour plus d'informations sur les différentes quadriques envisageables, se reporter en annexe A.

3.1.4 Le système de construction

Les principes de construction volumique

SGDLsoft est basé sur un système de modélisation logique des objets volumiques : l'utilisateur combine les primitives géométriques décrites à l'aide d'opérateurs volumiques comme l'ajout ou le prélèvement de matière. Toutefois, un grand nombre d'applications géométriques requièrent des processus de construction impliquant un certain nombre de contraintes (parallélisme, orthogonalité…) attachées à un ensemble de quadriques combinées volumiquement.

Dans la technologie SGDL, la prise en compte de ce genre de contraintes s'appuie sur le concept de configuration projective spatiale : schématiquement, on peut parler d'un squelette ou d'une ossature, conservant toutes ses propriétés d'incidence après transformations et déformations projectives et

sur lesquelles pourront venir se greffer les diverses composantes des objets volumiques voulus.

Pour gouverner et contraindre les objets volumiques, la technologie SGDL a retenu une des configurations projectives les plus simples, l'hexaèdre projectif. Comme le montre la figure 3.3, ses 12 arêtes se rencontrent 4 à 4 en 3 points particuliers et, selon la position respective de ces points, l'hexaèdre va se déformer tout en conservant la planéité de ses faces et ses relations d'incidence : ainsi, pour transformer cet hexaèdre en parallélépipède, il suffit d'envoyer les 3 points à l'infini dans des directions différentes. Et, pour définir parfaitement l'hexaèdre, il suffit de connaître ces 3 points, qu'on appellera points de fuite, et son centre [15, § G-II.5].

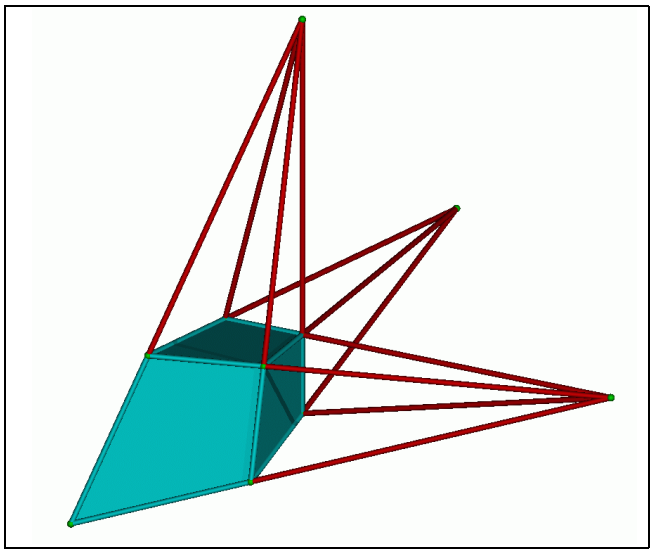


FIG. 3.3 – *L'hexaèdre projectif*

Cet hexaèdre projectif permet alors d'établir tout un vocabulaire de formes prédéfinies permettant de construire des objets volumiques précontraints projectivement. Ces formes prédéfinies dérivent bien évidemment de notre équation de la quadrique générale, mais elles ne sont plus construites en entrant 6 points de contrôle, mais en entrant les 5 points définissant son hexaèdre projectif, à savoir ses 4 points diagonaux (le point diagonal intérieur, ou centre de l'hexaèdre, et les 3 autres points diagonaux, les points de fuite de l'hexaèdre) ainsi que l'un de ses 6 sommets.

Il suffit alors de déformer cet hexaèdre pour déformer la forme prédéfinie, comme le montre bien l'exemple du demi-cône projectif de la figure 3.4 (pour les primitives pré-contraintes les plus utilisées, se reporter en annexe B).

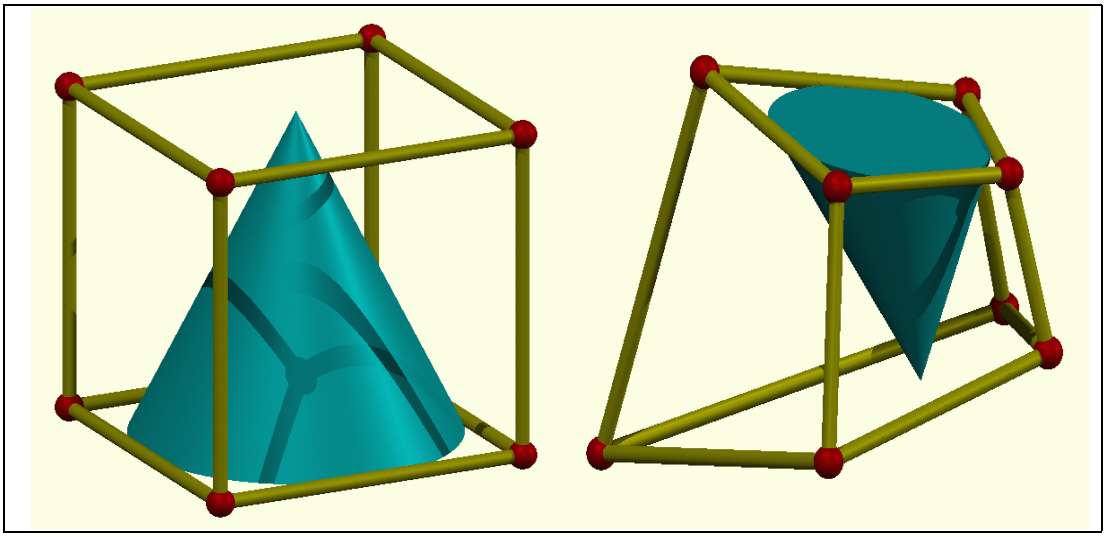


FIG. 3.4 – *Le demi-cône projectif et son tétraèdre de construction*

La logique des formes

Dans cette optique, le concept fondamental est celui de densité. Toute primitive géométrique de base partage en effet l'espace en 3 régions (extérieur, bord et intérieur), codées respectivement par les nombres entiers 0, 1 et 2. Ce codage ternaire initial de l'espace permet de connaître la densité de tout point dans l'espace par rapport à toute primitive élémentaire. L'extérieur de la quadrique sera différencié automatiquement par SGDLsoft qui saura reconnaître l'ensemble des points par lesquels on peut mener des tangentes réelles à la quadrique.

Il n'y a alors qu'un pas à franchir pour effectuer des critères de sélection numérique sur les régions de l'espace (en particulier en utilisant des opérateurs arithmétiques comme la multiplication avec des solides, ce qui revient à multiplier les densités et affecter ainsi les densités voulues aux volumes considérés) : par ce biais, il devient possible de filtrer la matière et d'extraire à partir d'une expression volumique plusieurs représentations filtrées d'un objet volumique initial (en particulier la "peau" du volume).

Muni d'un espace ainsi arithmétisé, il devient alors possible de combiner des primitives ou des volumes élémentaires à l'aide d'opérateurs arithmétiques définis par des fonctions primitives récursives telles que l'union, la soustraction ou l'intersection, permettant ainsi de faire le lien entre la géométrie projective et la logique au travers de l'arithmétique [15, § G-II.3].

3.1.5 Quelques considérations techniques

La distinction implicite/paramétrique

Le logiciel SGDLsoft s'appuie sur le développement de nouveaux formalismes implicites et paramétriques de surfaces algébriques projectives. Ces surfaces sont soumises à des contraintes projectives et contôlées par exemple par des points de passage, des points de contact avec des plans tangents...

Les “déplacements” de ces points permettent de modifier la nature euclidienne de la surface, donc son “aspect” physique. Toutefois, la connaissance projective de la surface permet à tout moment de connaître ses caractéristiques différentielles euclidiennes. Dans le contexte de la modélisation volumique, les équations implicites permettent de contrôler la position d'un point dans l'espace par rapport à la surface (le signe de la fonction implicite donne en effet cette position : intérieur, tangent, extérieur), alors que les équations paramétriques permettent de “circuler” sur la surface et donc de la mailler [15, § G-III.1].

Ainsi, l'équation implicite de la quadrique est une forme polynômiale du second degré. Ses 10 coefficients dépendent, dans la représentation interne SGDL, des coordonnées de 6 points de contrôle qui peuvent se mouvoir librement dans l'espace projectif, ce qui permet d'obtenir des déformations continues entre des quadriques propres (sphère, ellipsoïde…) et des quadriques dégénérées (cône, prisme…).

De la modélisation volumique au graphique

Les images calculées par les procédures particulières du langage SGDL appartiennent à une certaine classe d'images numériques générées par une technique de “ray-casting”. Cette technique permet de calculer pour chaque pixel de l'écran la partie de la scène volumique visible par un observateur situé dans l'espace. La qualité des images produites dépendra donc de la résolution générale de l'écran et en particulier de la taille de la fenêtre d'écran dans laquelle on désire afficher. Le temps de calcul des images dépendra par conséquent également de cette résolution, mais prendra aussi en compte deux nouveaux facteurs : la complexité de la scène volumique elle-même et le nombre de sources lumineuses disposées pour éclairer la scène.En prenant ces paramètres en compte et en les réglant, il est possible de parvenir à un bon rapport qualité d'image/temps de calcul.

Cette technique permet d'obtenir une qualité et une définition exactes des modèles géométriques représentés, tout en permettant un contrôle visuel optimal de l'environnement volumique créé. En effet, les objets et scènes volumiques générés avec SGDLsoft sont de réelles maquettes dont la visualisation ne peut être réalisée que par une technique de “lancer de rayon”. Il n'y a pas d'artifice, ce que l'on voit est exactement de qui est défini et stocké dans l'ordinateur [15, § P-III.2].

Cette technique détermine la visibilité d'une surface en traçant des rayons lumineux imaginaires depuis l'oeil de l'observateur jusqu'aux éléments de la scène, l'ordinateur calculant alors s'il y a ou non intersection des rayons avec ces éléments pour afficher ou non [4, Chap. 15]. Par conséquent, il est possible d'optimiser ces calculs en indiquant à l'ordinateur une zone précise où trouver des intersections, les calculs hors de cette zone étant abandonnés, d'où un gain de temps non négligeable, mais aussi un travail du programmeur plus conséquent pour déterminer ces zones. Le logiciel SGDLsoft permet cette optimisation en donnant la possibilité à l'utilisateur d'englober les différents éléments de ses scènes dans des enveloppes : on parle alors de **quadriques englobantes**, puisque la forme de base sous SGDLsoft est la quadrique (se reporter à la fonction DLmmx en annexe C).

La syntaxe

Il existe un certain nombre de conventions mnémoniques terminologiques indispensables pour bien comprendre le code SGDL. Voici les principales [15, § P-IV.2] :

- les types de paramètres :
  - V2 désigne un vecteur de 2 éléments (distance, angle, rapport...) : *(vector 1 2)*
  - V4 désigne un vecteur de 4 éléments (point, plan, couleur) : *(vector 1 2 3 4)*
  - V6 désigne un vecteur de 6 éléments (droite) : *(vector 1 2 3 4 5 6)*
  - V6V4 désigne un vecteur de 6 vecteurs de 4 éléments (mécanisme de construction d'une quadrique) : *(vector (vector 1 0 0 1) (vector -1 0 0 1) ... (vector 0 0 1 1))*
- les procédures : les noms des procédures ont tous 8 caractères, les 2 premiers en majuscules et indiquant les sous-librairies d'origine (SG, GD et SD), les 6 derniers en minuscules permettant de connaître les types de données géométriques (x pour plan, y pour droite, z pour plan, z2 pour quadrique, hxa pour hexaèdre projectif, con pour cône...).
- pour les principales fonctions utilisées, se reporter à l'annexe C.

La création et l'exécution d'un programme

La création d'un modèle volumique avec SGDLsoft comprend les phases de création de programme et d'exécution, ou interprétation, de programme.

Un éditeur de texte, comme par exemple X-Emacs sous Unix, est utilisé pour créer un ou plusieurs fichiers d'extension *.scm* contenant le programme qui définit la scène volumique.

Comme le montre la figure 3.5, l'exécution du programme s'effectue par application de l'interpréteur SGDLsoft aux fichiers de programme. Cette opération crée en sortie un fichier au format interne SGDL contenant la description complète de la scène volumique.

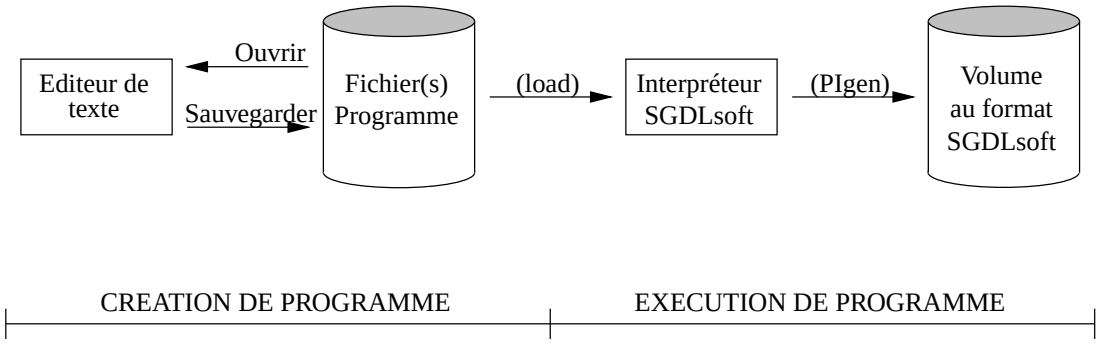


FIG. 3.5 – *Schéma de création d'une scène volumique sous SGDLsoft*

3.2 Le langage Scheme et sa programmation

La différence fondamentale entre SGDLsoft et un système traditionnel de modélisation tridimensionnelle est la possibilité de formuler explicitement à l'ordinateur la logique des processus de construction et de conception des objets à modéliser. C'est cette logique, étape par étape, qui va être mémorisée par la machine, ce qui va permettre à n'importe quel moment de reconstruire une copie de l'objet volumique programmé ou alors une variante lorsque ses composantes auront été préalablement paramétrées. Et toute cette puissance algorithmique de SGDLsoft n'est disponible que grâce à la complète intégration du langage SCHEME, dérivé de LISP, offrant ainsi au logiciel SGDLsoft un **environnement de programmation géométrique inconditionnelle**.

3.2.1 Introduction au langage Scheme

Programmer en Scheme consiste essentiellement à définir des fonctions : c'est un langage fonctionnel (on définit des fonctions et on les compose). Il n'y a pas de déclaration de type à faire pour les variables. Toutes les expressions Scheme sont écrites de façon préfixée, ce qui entraîne une utilisation intensive des parenthèses. On le constate facilement, on est loin d'un langage impératif comme le C, où le concept de base est l'*instruction*, dont le rôle est de provoquer un changement d'état mémoire de la machine : on parle d'*affectation*, là où Scheme fait de l'*évaluation* de *fonctions*).

L'utilisation des majuscules pour les 2 premiers caractères est une convention facilitant la lecture du code et l'identification des mots-clés propres à SGDLsoft et non au langage Scheme.

La syntaxe du langage Scheme est d'une grande simplicité et basée sur les notions illustrées par l'exemple qui suit :

```
(define EXadd1to (lambda (x) (+ x 1)))
```

On remarquera que ce morceau de code est présenté sans aller à la ligne, mais on peut parfaitement en changer la présentation si cela semble plus clair, comme par exemple :

```
(define EXadd1to
  (lambda (x)
    (+ x 1)
   )
 )
```

On vient ainsi de créer une procédure (ou fonction) Scheme, capable d'ajouter 1 à toute valeur numérique. Pour ajouter 1 à 1999, on tapera par exemple :

```
(EXadd1to 1999)
=> 2000
```

En fait, on associe un nom au choix (ici **EXadd1to**) à une procédure de calcul en utilisant le mot réservé **define** précédé d'une parenthèse ouvrante. La procédure de calcul elle-même est définie à

l'aide d'une forme spéciale spécifiée par le mot réservé **lambda** précédé d'une parenthèse ouvrante. Généralement, **lambda** s'utilise pour définir une procédure sans avoir recours à un nom. Nous aurions également pu créer une procédure sans utiliser **lambda** en spécifiant par exemple :

```
(define (EXajout1a x) (+ x 1))
```

l'intérêt d'utiliser **lambda** sera particulièrement évident dans des étapes de programmation plus avancées et il est bon de se familiariser le plus tôt possible avec son emploi [15, § P-III.1].

Pour plus de renseignements sur la syntaxe et les différentes procédures du langage Scheme, se reporter à [3, 16] ainsi qu'à l'annexe D.

3.2.2 La programmation inconditionnelle

Les diffrentes remarques évoquées dans le paragraphe 3.1 sur l'élimination des cas particuliers géométriques et numériques permettent d'envisager la recherche d'une programmation sans "IF", c'est-à-dire une programmation inconditionnelle.

L'aspect géométrique

En effet, le système de coordonnées adopté (*cf.* 3.1.2) permet de s'affranchir, d'un point de vue purement géométrique, des cas particuliers en gérant en particulier aussi bien les points à l'infini que les points plus conventionnels. Ainsi, dans le cas d'une droite et d'un plan non confondus, la fonction d'intersection retournera toujours un point projectif, dont la quatrième coordonnée sera égale à 0 si la droite est parallèle au plan ; si la droite et le plan sont confondus, alors toutes les coordonnées du point d'intersection seront nulles, indiquant dans ce cas que le point est indéterminé. On le voit bien, l'interprétation possible de cette indétermination ne nécessite ni message d'erreur, ni "IF" pour éviter ce message d'erreur, d'où une programmation plus concise et plus claire [15, §G-IV.4.1].

L'aspect numérique

Le traitement informatique de toute algorithmique géométrique utilise soit un formalisme symbolique, soit un formalisme numérique des données géométriques. Pour diverses raisons, incluant entre autres la rapidité et la moindre complexité des algorithmes, les logiciels de CAO traditionnels adoptent une approche numérique.

Un des problèmes majeurs du formalisme numérique par rapport au symbolique est la perte ou le manque de contrôle de la précision des calculs, qui peut entraîner des solutions dénuées de toute signification logique. Une des raisons essentielles de cette problématique est la représentation flottante des nombres en machine qui conduit à l'utilisation d'une arithmétique des ordinateurs qui diffèrent en particulier de l'arithmétique traditionnelle, dès que la taille des nombres manipulés échappe à la place mémoire qui leur est affectée. Il s'établit alors un compromis précision-espace mémoire, acceptable dans les logiciels de DAO, mais fortement incompatible avec la plupart des problèmes volumiques ou les applications géométriques liées à l'intelligence artificielle.

Un compromis intéressant entre le formalisme symbolique et le formalisme numérique flottant est le formalisme numérique entier : en effet, tant que les calculs portent sur des nombres rationnels, le système de coordonnées projectives permet une algorithmique en nombres entiers. La précision des calculs est alors totale et les résultats obtenus comparables à ceux obtenus par le calcul symbolique. Lorsque les calculs portent sur des irrationnels, le passage à une numération en nombres entiers peut se faire par une approximation en rationnels ; les résultats obtenus n'ont plus la même précision qu'en calcul symbolique lorsque des simplifications exactes sur des irrationnels sont possibles, mais par rapport à une numération en nombres flottants, les endroits dans le code où les approximations réalisées sont parfaitement localisées car dépendant du programmeur et non du système : le contrôle de la cohérence entre l'algorithmique et la numération reste possible [15, §G-IV.4.2].

3.2.3 Les styles de programmation

Le langage Scheme est un langage de **programmation fonctionnelle et récursive**, c'est pourquoi il peut être intéressant de s'attarder sur les caractéristiques d'un tel type de programmation, en particulier par rapport à la programmation bien plus conventionnelle qu'est la programmation impérative et itérative.

Programmations impérative et fonctionnelle

La programmation impérative est en fait le modèle le plus commun de programmation, c'est en particulier le modèle dont s'inspirent les langages (dits impératifs) comme le Fortran ou bien encore le C. Et contrairement à ce type de programmation, la programmation fonctionnelle conserve le comportement mathématique de la variable au niveau informatique. Elle obéit en effet à un modèle théorique de description algorithmique, basé sur les concepts de récursivité et de λ-calcul.

Dans le cas de la programmation impérative, le programme gère la cinématique d'exécution du programme par la machine, il programme en fait la machine pour automatiser l'exécution du programme. En programmation fonctionnelle, seul le déroulement logique et fonctionnel est pris en compte par le programmeur. Cela est rendu possible par une syntaxe complètement tournée vers des notions de mathématiques pures comme l'application ou la composition de fonctions, ainsi que par la simple impossibilité d'affecter, de gérer la mémoire de la machine ou bien de piloter l'ordre d'exécution des instructions [15, §G-IV.5.1].

Contrairement à la programmation impérative qui décrit un processus physique du déroulement d'un algorithme, la programmation fonctionnelle décrit le processus logique de ce déroulement. C'est le style de programmation que nous avons essayé d'adopter.

Programmations itérative et récursive

La modélisation volumique d'objets ou de scènes complexes comprend parfois des répétitions d'objets semblables ou de variantes de ces objets. Dans le cadre de la programmation, on est par conséquent amené à utiliser des instructions permettant de décrire des mécanismes répétitifs. Ces mécanismes peuvent être principalement itératifs ou bien récursifs.

Dans le cas de l'itération, des mécanismes de "boucle" mettent à jour un nombre fini de variables d'état qui doivent être elles-mêmes mises à jour à chaque répétition du processus de calcul. A tout moment l'arrêt du processus permettra de vérifier l'état général de ces variables et du processus lui-même.

Dans le cas de la récursivité, un processus est défini par un ensemble de niveaux de calculs dépendant les uns des autres en "cascade". L'état général d'un processus à un certain moment est par conséquent toujours différé car les variables utilisées passent par des phases transitoires. Dans certains cas, certains interpréteurs comme Scheme permettent de décrire des processus itératifs par des mécanismes ou procédures récursives. L'intérêt pour le programmeur est alors d'éviter les mécanismes de contrôle itératifs qui sont non-fonctionnels et souvent délicats à vérifier [15, §G-IV.5.2].

Pour conclure, schématiquement, dans un mécanisme itératif, on spécifie la manière dont un volume élémentaire est répété pour constituer un volume plus complexe, alors qu'un processus récursif utilise un volume élémentaire pour générer un volume plus complexe, qui à son tour va servir de point de départ pour la génération d'un nouveau volume, et ce autant de fois que le programmeur le désirera [14, p. 226-227].

3.3 Les autres outils

Tout au long de notre projet, nous avons travaillé avec SGDL soft sur des stations de travail SUN, équipées de processeurs Spark 5 et de 32 Mo de RAM, mais aussi sur des PC sous Linux (distributions RedHat 5.2, puis 6.0), équipés de processeurs Intel Pentium III cadencé à 450 MHz et de 128 Mo de RAM.

De plus, pour créer nos animations, nous avons utilisé le logiciel QuickTime Pro 4.0, donc toutes les animations peuvent être visualisées avec le shareware QuickTime Player version 4 (en distribution libre sur Internet).

Enfin, pour la rédaction de notre rapport, nous avons travaillé avec le logiciel de formatage typographique LaTeX [1, 5, 7, 9, 13, 18] (LaTeX 2_ε, sous MlTeX version 2.2), afin d'obtenir un rapport de qualité professionnelle, qui soit portable sur tous les environnements, à condition de disposer du shareware Acrobat Reader (lui aussi en distribution libre sur Internet). La conversion du fichier natif LaTeX en format Acrobat (extension *.pdf*) a été possible grâce au logiciel Adobe Distiller.

Chapitre 4

L'approche "taillage"

4.1 Présentation

Au niveau mathématique, comme on l'a déjà vu en 2.3, les engrenages hélicoïdaux soulèvent énormément de difficultés au niveau de la définition de la surface des flancs de denture, des hélicoïdes en développante de cercle. C'est pourquoi nous avons essayé dans un premier temps de nous affranchir de cet aspect en envisageant une approche radicalement différente basée sur la simulation du mode d'élaboration de l'engrenage.

A ce niveau de l'étude, il devient impératif de rappeler quelques notions élémentaires à propos de l'élaboration des engrenages. Il existe en effet plusieurs modes de fabrication des engrenages, que l'on peut classifier en deux grands groupes, selon que les profils actifs sont obtenus :

- sans enlèvement de matière (sans copeaux) :
  - moulage sable,
  - moulage sous pression,
  - forgeage / estampage,
  - barre étirée au profil de l'engrenage,
  - découpage ;
- avec enlèvement de matière (avec copeaux) :
  - taillage par fraise de forme (fraise doigt ou fraise "au module"),
  - taillage par génération : l'outil génère, engendre les profils actifs ; le profil est obtenu par l'enveloppe des positions successives de l'outil, c'est le procédé le plus utilisé dès que l'on désire une qualité correcte :
    - taillage à l'outil crémaillère (procédés Maag ou Sunderand-Rollet),
    - taillage à l'outil pignon (procédé Fellow),
    - taillage á la fraise-mère.

Pour un engrenage hélicoïdal, tous ces procédés ne sont bien évidemment pas applicables, et nous avons choisi arbitrairement de ne nous intéresser qu'au cas du taillage à l'outil crémaillère, choix dû aussi au fait que la crémaillère est l'outil le plus simple à représenter de manière informatique.

Dans ces conditions, une nouvelle problématique s'offrait à nous : comment mettre en oeuvre une telle approche tout en respectant les conditions élémentaires de roulement sans glissement au niveau des primitifs, de géométrie de l'outil utilisé et de cinématique associée, ainsi qu'en utilisant au mieux les potentialités du logiciel SGDLsoft ?

4.2 La génération de l'outil crémaillère

Une des principales propriétés des engrenages en développante est que l'engrenage devient une crémaillère, avec des flancs rectilignes dans un plan perpendiculaire à l'axe de rotation de l'engrenage complémentaire, quand le nombre de dents tend vers l'infini.

Cette crémaillère peut engrèner avec n'importe quel autre engrenage en développante, pourvu que le module et l'angle d'hélice soient les mêmes , avec des lignes de contact rectilignes, et uniquement si les flancs de la crémaillère sont plans.

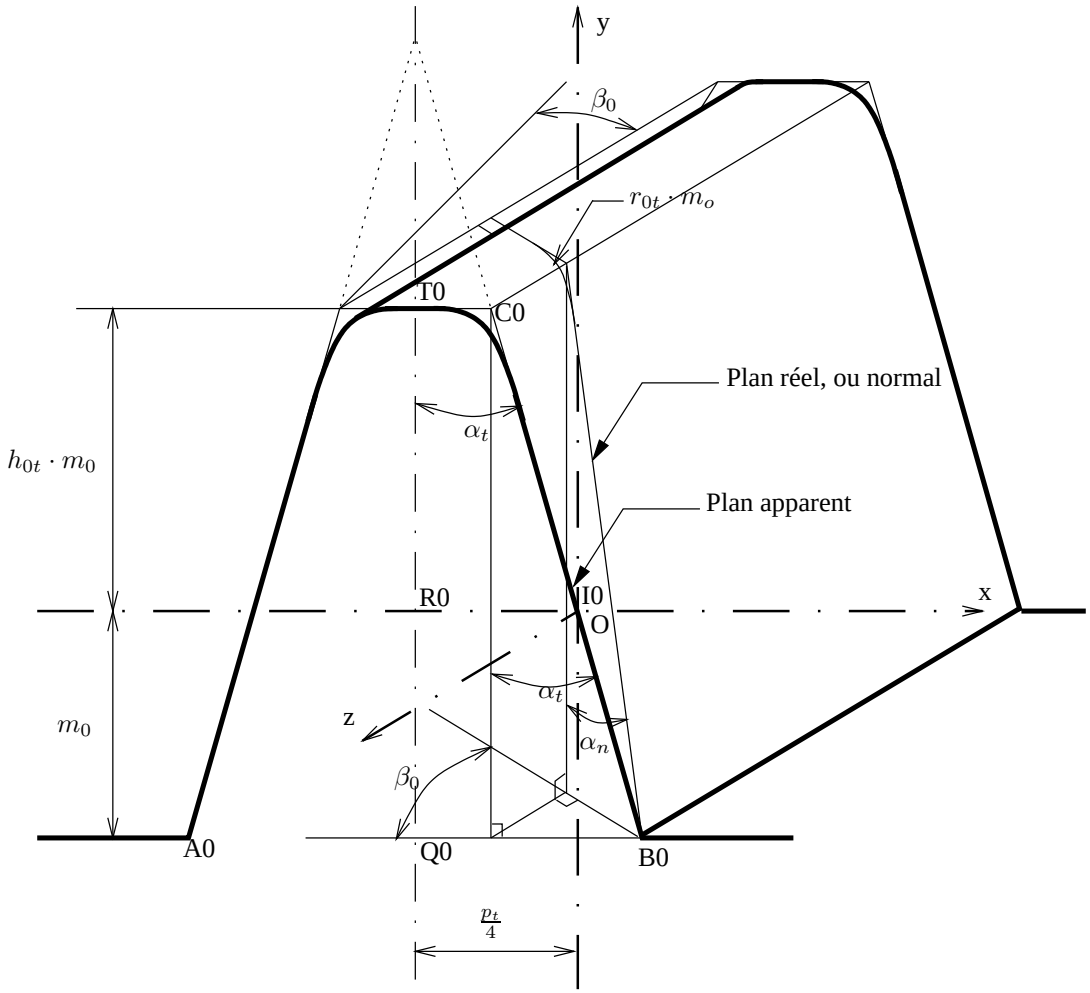


FIG. 4.1 – *Géométrie de la dent de la crémaillère de taillage*

Cette propriété rend possible la fabrication d'engrenages en utilisant comme outil de taille une crémaillère avec des flancs plans. La figure 3.2 montre la géométrie d'une telle crémaillère, avec un angle de pression réel α_n, un angle d'hélice β_0 (*cf.* les pages de notations au début du rapport).

A propos de l'angle d'hélice, il est bon de rappeler qu'en pratique, pour fabriquer un engrenage hélicoïdal, on utilise en général une crémaillère avec un angle d'hélice nul qu'on préfère incliner de l'angle d'hélice voulu par rapport à l'axe des x.

Pour notre approche, nous avons délibérément choisi de prendre une crémaillère avec un angle d'hélice en paramètre, donc sans jouer sur l'inclinaison de la crémaillère elle-même mais plutôt en jouant sur l'inclinaison de ses dents, afin de nous placer directement dans un contexte projectif.

Pour représenter cette crémaillère, nous avons dans un premier temps travaillé en analytique sur les figures 4.1 et 4.2 en calculant toutes les équations dans l'espace des points nécessaires à la construction de cette crémaillère (pour avoir le détail des calculs, se reporter en annexe E). Ce fut une erreur de notre part due surtout à notre méconnaissance des potentialités projectives du logiciel SGDLsoft (il faut bien se rappeler que ce travail a été réalisé dans les 3 premières semaines qui ont suivi notre arrivée au sein du GRCAO : *cf.* planning en figure 2.1) : le code développé a alors été repensé entièrement en faisant appel cette fois aux fonctionnalités projectives, limitant ainsi énormément les calculs (se reporter en annexe F pour le détail commenté du code de génération de la dent de la crémaillère).

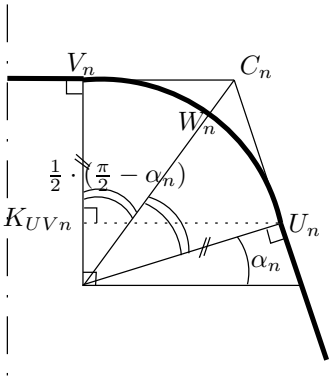


FIG. 4.2 – *Vue dans le plan P_n (plan normal passant par I0) de l'arrondi en tête de dent*

En particulier, la difficulté essentielle résidait au niveau de l'arrondi en tête de dent : en effet, on a effectivement un raccord circulaire à ce niveau, mais uniquement si l'on se place dans un plan normal. Dans le plan apparent, on verrait plutôt un raccord elliptique. Et SGDLsoft offre la particularité de pouvoir faire relativement facilement ce type de raccord projectif, pourvu qu'on connaisse les 2 points de tangence et l'intersection des tangentes en ces points. Or on peut facilement déterminer le point C_n, intersection des tangentes en question, et en déduire assez aisément les points de tangence U_n et V_n par le biais de calculs projectifs (projections dans différents plans connus, cf. annexe E) ; dès lors on se retrouve dans une configuration déjà étudiée auparavant au paragraphe 3.1.4 : le triangle de construction d'une conique. Ainsi, ces 3 points suffisent à définir la conique de raccordement entre U_n et V_n (ici un cercle), en passant par l'intermédiaire d'un point de passage déterminé par la fonction SGDL SGy2_cir. Pour obtenir le raccordement sur toute la dent, il suffit d'envoyer l'autre pôle

(intersection des tangentes) à l'infini dans la direction de β_0. C'est ce qu'illustre la figure 4.3.

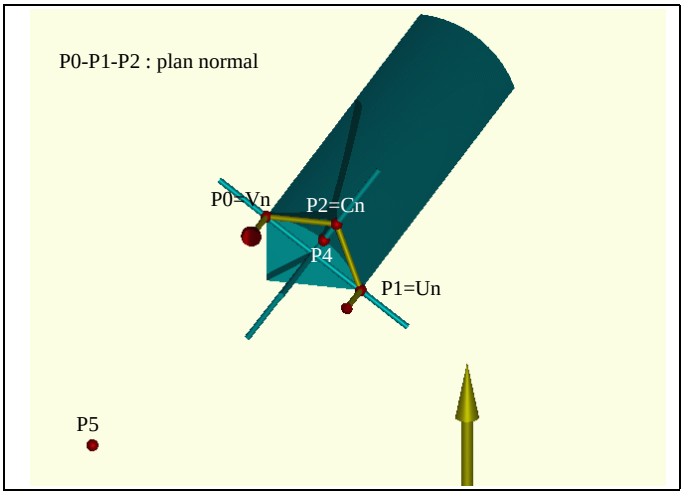


FIG. 4.3 – *Points de contrôle de la quadrique définissant l'arrondi en tête de dent*

Ainsi, de manière schématique, à partir de points de construction précis définis de manière projective (les points définis sur les 2 figures précédentes), on commence par construire une portion de prisme incluant le paramètre d'hélice β_0 à partir de la forme générale d'une quadrique, en imposant de manière particulière les points de contrôle de la quadrique. De la même manière, on génère une "boîte" parallélépipédique incluant elle aussi le paramètre d'hélice, de même que les cylindres qui viennent ensuite pour générer les arrondis (dans le plan normal) en tête de dent de la crémaillère. L'union de ces différents solides donne une dent de la crémaillère, c'est ce qu'illustre la figure 4.4. Pour avoir la crémaillère en entier, il suffit d'espacer convenablement les dents (pas apparent p_t selon les x) et de leur ajouter un socle parallélépipédique.

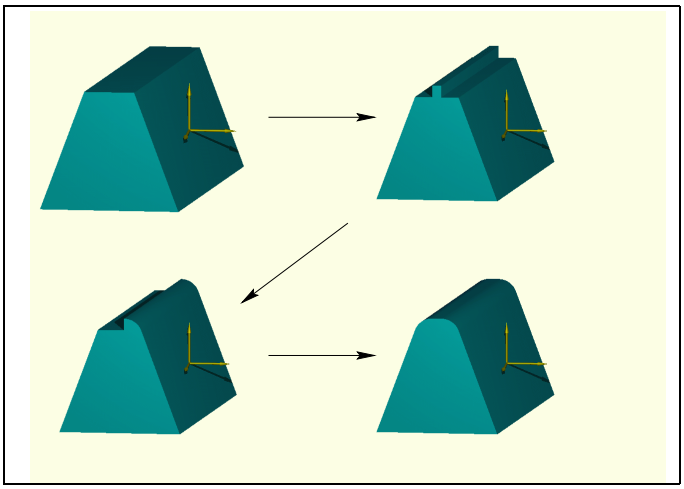


FIG. 4.4 – *Schéma de construction d'une dent*

4.3 La représentation des engrenages

A présent, nous disposons de la crémaillère de taillage, nous sommes donc théoriquement en mesure de générer un engrenage : en effet, pour cela, il suffit d'appliquer des lois cinematiques adéquates entre la crémaillère et une roue de largeur égale à la largeur b de l'engrenage voulu et de diamètre minimal celui de tête, à savoir le diamètre primitif, fonction du nombre de dents voulu, auquel on ajoute 2 fois le module m_0 (on se place dans l'hypothèse où l'on n'a pas de déport).

Voilà pour la théorie, mais comment mettre cela concrètement en oeuvre ? En fait plusieurs possibilités s'offraient à nous, mais avant toute chose, il fallait prendre soin bien évidemment de positionner correctement la crémaillère par rapport au bloc cylindrique (correspondance des primitifs, placement du point I0 à l'origine du repère), comme le montre la figure 4.5.

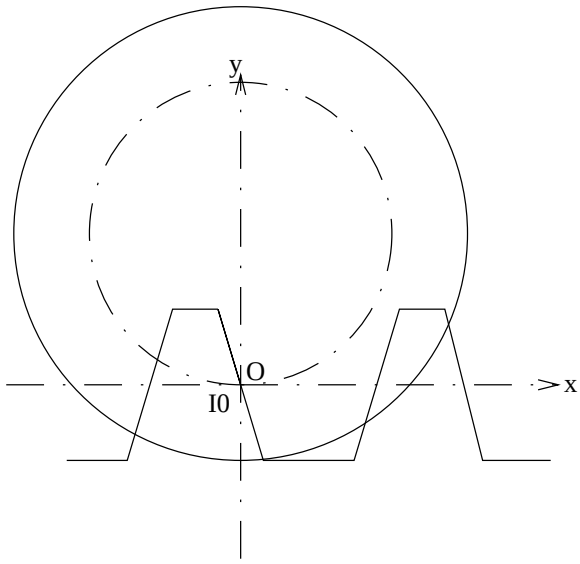


FIG. 4.5 – *Positionnement initial de la crémaillère*

4.3.1 Rotation de la roue + translation de la crémaillère

Dans cette optique, le principe est d'attibuer un incrément de déplacement angulaire sur la roue qui détermine la longueur du pas de déplacement linéaire de la crémaillère (le pas angulaire corres pond au pas linéaire : c'est la condition de roulement sans glissement) ; il suffit alors à chaque itération de soustraire à la roue la crémaillere (ce qui revient à enlever à la roue la matière en superposition avec celle de la crémaillère) et d'appliquer l'itération à la roue ainsi modifiée : c'est une boucle récursive, on utilise à chaque itération le volume auparavant modifié. La portion de code ci-dessous correspond à la formulation définitive de cette boucle sous SGDL :

```
(let boucle
    ((roue cylin)
     (i 0))
  (if (<= i (- k 1))
      (boucle
       (taille roue cremaillere i mmx-cremaillere-taillage)
```

```
      (+ i 1))
     (tracer roue cremaillere mmx-roue mmx-cremaillere)
     ))
```

C'est une structure "named let" (*cf.* annexe D), pour laquelle on initialise 2 paramètres `roue` (ce volume est initialisé avec le cylindre `cylin`) et `i` (cet incrément est initialisé à 0) ; ensuite cette boucle lance la procédure de taillage `taille` qui a pour argument les volumes `roue` et `cremaillere`, l'incrément et la quadrique englobante de la crémaillère jusqu'à ce que l'incrément atteigne le nombre de divisions angulaires souhaité `k` ; une fois ce dernier atteint, la boucle bascule sur le mode de tracé du volume final, c'est-à-dire notre engrenage, en lançant la procédure `tracer`. Pour plus de détails sur les différentes procédures mises en jeu ou les différents paramètres entrés en arguments de ces procédures, se reporter en annexe F.

4.3.2 Rotation + translation de la crémaillère

Cette fois, la roue reste fixe : on fait tourner la crémaillère autour de la roue, mais pour que le positionnement relatif entre les deux soit correct, il faut auparavant la translater d'une quantité égale à la longueur de l'arc qu'on lui fait parcourir depuis sa position d'origine, soit le produit du nombre d'itérations déjà effectuées par la valeur de l'incrément par le rayon primitif. On stocke l'ensemble de ces crémaillères ainsi déplacées dans une liste qu'on soustrait de manière récursive à la roue, c'est-à-dire qu'on commence par enlever à la roue la crémaillère dans sa position initiale, on applique alors la première itération à la crémaillère, qui se retrouve ainsi déplacée de la valeur correspondante au premier incrément, et on enlève cette crémaillère à la roue précédemment modifiée, et ainsi de suite… La boucle développée dans le code ci-dessous correspond à la structure algorithmique décrite juste avant :

```
(let boucle
    ((list_crem (list))
     (i 0))
  (if (< i k)
      (boucle
       (cons
        (translation-rotation (* i pas_ang) cremaillere)
        list_crem)
       (+ i 1))
      (tracer
       (apply DLdif (cons cylin list_crem))
       cremaillere  mmx-roue mmx-cremaillere)
      ))
```

Là aussi la boucle a une structure de "named let" pour laquelle dans un premier temps on initialise les paramètres de la boucle, à savoir la liste `list_crem`, à l'intérieur de laquelle on va venir stocker nos différentes crémaillères et l'incrément ; ensuite la boucle construit la liste des crémaillères à chaque itération (les crémaillères sont déplacées grâce à la procédure translation+rotation) : chaque nouvelle crémaillère est ajoutée au début de la liste `list_crem`. Pour plus de détails, se reporter aussi en annexe F.

4.3.3 Rotation + translation de la roue

Cette dernière optique envisage une crémaillère fixe sur laquelle viendrait faire rouler sans glisser la roue. Cette apporche est aussi viable que les autres mais posent autrement plus de problèmes en ce qui concerne le rendu de la scène volumique décrite : en effet, comme on l'a vu au paragraphe 3.1.5, le système de "ray-casting" suppose de définir un observateur et un objet-cible ; or ici l'objet-cible serait la roue, ici ayant un mouvement de translation en plus de la rotation sur elle-même, c'est-à-dire que la cible ne serait plus fixe, ce qui pose des problèmes de suivi de la roue en mouvement pour la représentation.

4.4 Résultats et conclusions

La mise en place de l'approche orientée sur le taillage de l'engrenage a globalement été menée rapidement, les problèmes d'optimisation de code étant les principaux problèmes que nous ayons rencontrés, mais nous y reviendrons au chapitre 6. Suite aux problèmes que soulève la génération avec la crémaillère fixe et la roue en rotation/translation, nous avons rapidement abandonné cette optique pour ne nous concentrer que sur les 2 autres. Il est à noter que celles-ci d'ailleurs sont à peu près équivalentes en termes de temps de calculs ; aussi, puisque l'aspect taillage nous intéressait plus particulièrement, il nous a semblé plus opportun de finaliser notre code avec l'algorithme translation de la crémaillère/rotation de la roue, plus proche de la réalité et de la finalité de notre étude que l'algorithme de stockage de l'enveloppe des crémaillères dans une liste.

Cet approche nous a permis de générer rapidement des engrenages à partir de quelques paramètres mécaniques seulement, comme le module m_0, l'angle de pression α_0, l'angle d'hélice β_0, la largeur b et le nombre de dents Z. De plus, le code utilise pleinement les capacités projectives du logiciel SGDLsoft : ainsi, pour tracer un engrenage droit, il suffit de mettre le paramètre β_0 à 0 et tout le code suit derrière. De même, le pied de dent au niveau de l'engrenage, souvent à l'origine de bien des problèmes pour les outils de représentation des engrenages, qui ont une approche plus analytique, ne pose ici aucun problème puisqu'entièrement paramétré au niveau de la crémaillère et non au niveau de l'engrenage.

Par ailleurs, la précision est contrôlée par le pas angulaire (en fait, ce n'est pas directement le pas angulaire qu'on paramètre, mais le nombre de divisions que l'on réalise sur 360^o , ce qui permet d'avoir de véritables divisions entières et non un incrément angulaire pouvant ne pas tomber juste...) : ainsi, à titre indicatif, pour un pas équivalent à un pas angulaire de 1^o , quelle que soit les machines à notre disposition sur lesquelles on lançait le calcul, le temps nécessaire à l'affichage était inférieur à 10 mn.

Enfin, avec cette approche, il est possible d'insérer facilement l'engrenage dans une scène volumique complexe, de l'engrenage seul à plusieurs engrenages engrènant ensemble en passant par un engrenage et sa crémaillère de taillage : ce n'est qu'un problème de positionnement adéquat des différents éléments composant la scène volumique recherchée.

Par contre, cette approche permet de n'aborder que l'aspect représentation des engrenages, elle n'autorise pour le moment aucune sortie mathématique intéressante pour un post-traitement sur les

engrenages (notices de calculs. . .).

En conclusion, en adoptant une démarche assez originale fondée plus sur le processus de génération de l'engrenage que sur l'engrenage lui-même, nous avons montré qu'il était possible grâce au logiciel SGDLsoft de représenter rapidement des engrenages, y compris des engrenages aussi complexes que les engrenages hélicoïdaux avec leurs pieds de dent, en nous privant toutefois de toute sortie mathématique exploitable par la suite.

Chapitre 5

L'approche mathématique

5.1 Présentation

Après avoir travaillé sur la modélisation du procédé d'obtention des engrenages, nous nous sommes ensuite attachés à la problématique liée aux mathématiques. En effet, il est aussi possible de générer des engrenages à partir de leurs équations mathématiques. Ces équations sont données sous formes paramétriques ou polaires : en effet, celles-ci combinent des développantes de cercles et des hélices, l'écriture sous forme implicite d'une hélicoïde est donc délicate. En fait, l'hélice est une courbe transcendantale, autrement dit c'est une fonction polynômiale de degré infini, ce qui impose d'un point de vue informatique de faire des approximations, soit en utilisant des fonctions trigonométriques, elles-mêmes connues avec une certaine précision, soit en se limitant à un certain degré pour les polynômes.

Le problème revient donc à utiliser des équations paramétriques sur un modeleur volumique ayant pour primitives géométriques les quadriques, qui elles sont connues à partir d'équations implicites. Ce passage nécessitera inévitablement un choix dans les approximations des courbes et des surfaces.

5.2 Les choix techniques

5.2.1 Le modèle mathématique

Les différents modèles

Notre choix a été motivé par notre recherche bibliographique : en fait, il existe assez peu de documents traitant la problématique des engrenages de manière mathématique et informatique. Nous avons étudié deux équations simples avant d'opter pour le modèle de M. Eitel T. Lauria [10]. Ces équations sont planes, voici leurs expressions :

– pour l'équation polaire (le lecteur s'aidera de la figure 5.1, tirée du roulement sans glissement en I) :

$$\begin{cases} \theta = \frac{IM}{r_b} - t = \tan t - t, \, fonction\ involute \\ \rho = \frac{r_b}{\cos t} \end{cases}$$

Ici, tout le problème réside en fait dans cette fonction involute : en effet, pour pouvoir tracer la courbe polaire fonction de t, il faut déterminer la fonction réciproque de l'involute et ce n'est possible qu'avec des tables d'involutes ou des algorithmes informatiques de convergence. Pour notre part, nous avons utilisé celui de NEWTON-RAPHSON, étudié plus en détail dans le chapitre 6.

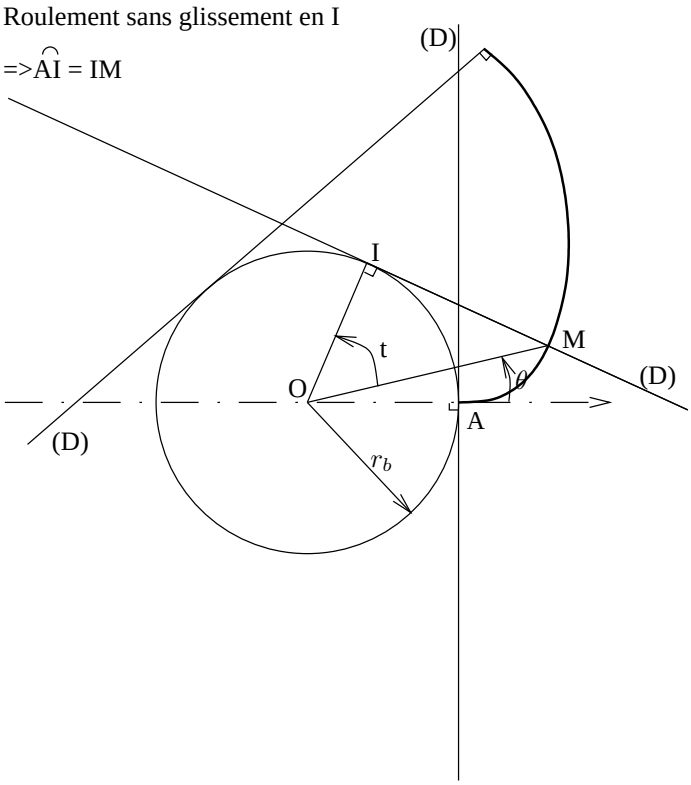


FIG. 5.1 – *Développante de cercle*

– pour l'équation paramétrique , tirée du paramétrage classique du cercle en mathématiques :

$$\begin{cases} x(u) = r_b \cdot (\cos u + u \cdot \sin u) \\ y(u) = r_b \cdot (\sin u - u \cdot \cos u) \end{cases}$$

Le modèle de M. LAURIA nous est apparu comme plus complet, notamment parce qu'il permet de gérer le profil actif des dents, il effectue en outre le calcul de paramètres mécaniques intéressants.

Le modèle de Eitel T. Lauria

Le modèle de M. LAURIA permet de générer un maillage surfacique représentant une surface d'hélicoïde à partir de deux paramètres ϵ et Z. Lorsqu'on travaille à Z constant, l'équation se réduit à celle d'une développante de cercle, donc finalement on peut travailler sur ce maillage en étudiant les profils relatifs à des sections d'axe Z.

De plus, ce modèle propose le calcul de nombreux angles, rayons, et autres paramètres mécaniques tels que α'_t, $r'1$, s_{b1}, g_f , g_a, ϵ_γ... Il faut enfin noter un détail important : ce paramétrage n'est pas uniforme lorsqu'on fait varier ϵ.

5.2.2 Les approximations nécessaires

Au cours des premières réunions avec M. ROTGÉ, il a été décidé d'adopter une démarche progressive en termes de difficulté et de qualité des résultats obtenus. Cela s'est traduit par le schéma de travail suivant :

- dans un premier temps, effectuer une approximation de type linéaire, en utilisant uniquement des primitives géométriques linéaires pour la développante de cercle et des volumes à faces planes pour l'approximation de l'hélicoïde ;
- dans un second temps, travailler sur la problématique d'une approximation quadratique, où la montée en degré des primitives géométriques permettra une meilleure approximation, que ce soit au niveau de la développante de cercle pour l'aspect 2D, ou au niveau de l'hélicoïde pour la construction de la dent complète.

5.3 Approximation linéaire

5.3.1 Approximation du profil en développante de cercle par des segments

Cette partie s'intéresse à un problème plan, celui de l'approximation d'une développante de cercle. Au début de nos travaux, la manière la plus simple de chercher à approcher la développante de cercle est évidemment celle utilisant des segments, nous avons donc choisi de faire varier ϵ linéairement, même si cela génère une répartition des points non uniforme.

Après avoir modélisé le profil de la développante, il faut obtenir une surface, et pour construire un profil de dent complet, il faut connaître l'axe de symétrie de la dent. Dans cette optique, nous avons approfondi les travaux de M. LAURIA, en recherchant la dépendance de cette axe de symétrie par rapport à Z : en effet, la plupart des figures d'engrenages sont planes et se rapportent à un axe fixe. Nos calculs pour positionner l'axe de la dent dans l'espace donnent l'expression suivante (*cf.* figure 5.2) :

$$\Psi(Z) = -\frac{s'_1}{2 \cdot r'_1} + \frac{g_\beta \cdot Z}{b \cdot r_{b1}}$$

Ainsi nous avons construit une succession de trapèzes symétriques par rapport à l'axe de la dent. Pour plus de détail sur la construction géométrique, se reporter au code en annexe G.

5.3.2 Approximation de l'hélicoïde à partir d'une succession de boîtes à base trapézoïdale

Pour construire un volume à partir de l'équation surfacique d'un profil de dent, nous avons travaillé par section sur l'axe Z, tout le problème revient donc à faire passer un volume par deux sections consécutives.

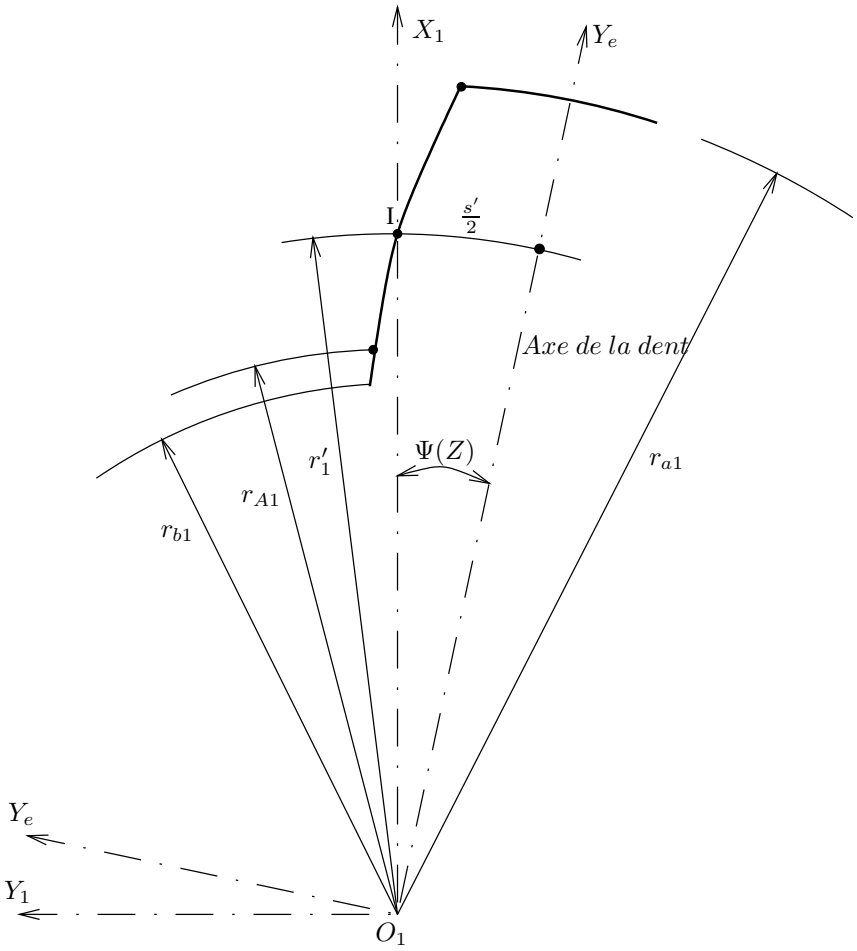


FIG. 5.2 – *Forme de la développante dans une section Z = 0*

Dans le cas d'un engrenage droit, les profils sont alignés suivant l'axe Z, le volume correspondant est donc obtenu par extrusion suivant cet axe. Par contre, dans le cas d'un engrenage hélicoïdal, les profils ne sont pas alignés : en effet, entre deux profils, il y a eu une opération de translation et de rotation ; or, comme dans nos hypothèses nous ne devions utiliser que des volumes à surface plane, il était impossible de joindre ces points appartenant à une surface gauche.

En conclusion, la discontinuité de matière au niveau des raccords entre les différentes sections était inévitable, comme le montre la figure 5.3. Le choix a donc été de générer des dents élémentaires obtenus par extrusion sur Z, ce qui correspond, en géométrie projective, à envoyer un pôle à l'infini suivant la direction Z.

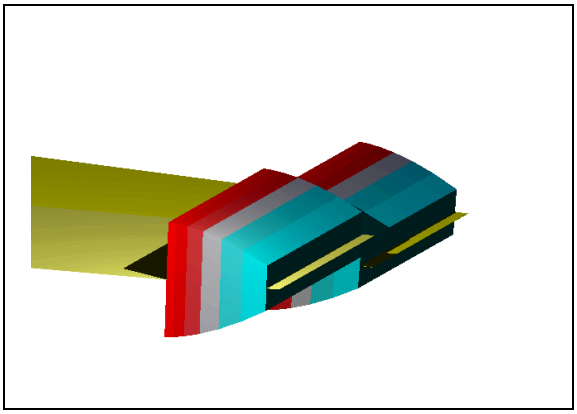


FIG. 5.3 – *Approximation linéaire de la denture*

5.3.3 Les résultats de l'approche linéaire

La qualité des premiers résultats obtenus était prévisible : si, pour un engrenage droit, les résultats étaient corrects en termes de qualité visuelle et de nombre de quadriques calculées, dans le cas d'un engrenage hélicoïdal, la discontinuité de matière au niveau des raccords impose le calcul de très nombreuses quadriques pour approcher précisément la dent, comme l'illustre la figure 5.4).

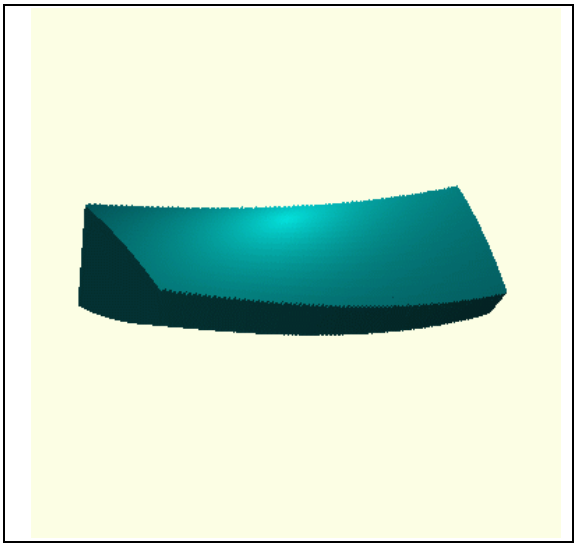


FIG. 5.4 – *Résultat final de l'approximation linéaire de la denture hélicoïdale*

Nous n'avons pas poussé les recherches plus loin dans ce type d'approximation. En effet, cette première étape a eu le mérite de bien poser les bases de la problématique, nous pouvions donc passer rapidement à l'approximation quadratique, beaucoup plus intéressante à nos yeux.

5.4 Approximation quadratique

5.4.1 Approximation de la développante de cercle par une conique

Introduction

Dans notre étude, nous souhaitions nous affranchir du problème de non uniformité du paramètre ϵ, et dans cette optique nous avons choisi d'approximer la développante par une portion de conique seulement, en utilisant au mieux les potentialités de SGDLsoft.

Ainsi, nous avons décidé de ne prendre que les points extrêmes du profil actif pour déterminer les points de tangence de la conique et son triangle de construction (*cf.* paragraphe 3.1.3 et figure 5.5). Ces points dépendent de deux valeurs extrêmes de ϵ, nommées ϵ_f et ϵ_a ; ces valeurs, fonctions du paramètre Z, sont recalculées pour chaque section sur l'axe Z.

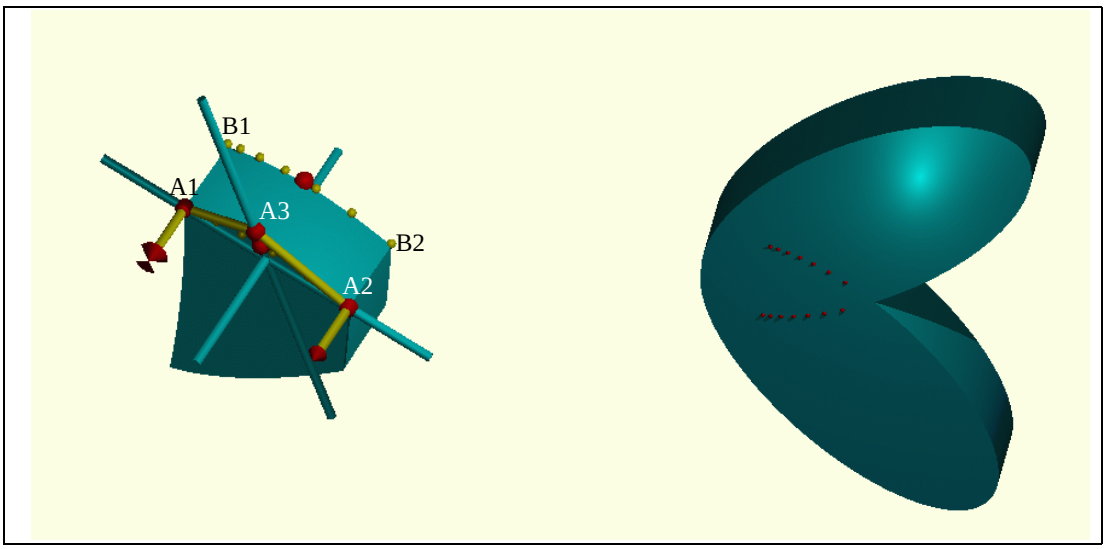


FIG. 5.5 – *Figure de construction de l'approximation quadratique de la développante*

Le calcul du pôle

Le pôle étant, par définition, l'intersection des tangentes, nous avons calculé analytiquement les tangentes aux points extrêmes, les travaux de M. LAURIA [10] n'ayant pas fourni ce calcul. Les équations explicitant la tangente à la développante de cercle en tout point sont donc présentées ci-dessous, mais juste avant se trouve un rappel de l'équation paramétrique de M. LAURIA définissant l'hélicoïde (*cf.* figure 5.2) :

$$\begin{cases} X_1(\epsilon,Z) = Q_{xz} \cdot Z + Q_{x\epsilon} \cdot \epsilon + Q_x \\ Y_1(\epsilon,Z) = Q_{yz} \cdot Z + Q_{y\epsilon} \cdot \epsilon + Q_y \\ Z_1(Z) = Z \end{cases}$$

où les différents facteurs Q dépendent de λ_1, donc de ϵ, comme le prouvent les équations suivantes :

$$\begin{cases} Q_{xz} = K_1 \cdot \cos\lambda_1(\epsilon) + K_2 \cdot \sin\lambda_1(\epsilon) \\ Q_{x\epsilon} = K_3 \cdot \cos\lambda_1(\epsilon) + K_4 \cdot \sin\lambda_1(\epsilon) \\ Qx = r_1' \cdot \cos\lambda_1(\epsilon) + K_5 \cdot \cos\lambda_1(\epsilon) + K_6 \cdot \sin\lambda_1(\epsilon) \end{cases}$$

$$\begin{cases} Q_{yz} = K_2 \cdot \cos\lambda_1(\epsilon) - K_1 \cdot \sin\lambda_1(\epsilon) \\ Q_{y\epsilon} = K_4 \cdot \cos\lambda_1(\epsilon) - K_3 \cdot \sin\lambda_1(\epsilon) \\ Qy = -r_1' \cdot \sin\lambda_1(\epsilon) - K_5 \cdot \sin\lambda_1(\epsilon) + K_6 \cdot \cos\lambda_1(\epsilon) \end{cases}$$

où les différents K sont des constantes liées aux grandeurs mécaniques et avec :

$$\epsilon = \frac{r_{b1} \cdot \epsilon_\gamma}{g_f + g_a + g_\beta} \cdot \lambda_1(\epsilon) + \epsilon_I$$

Pour obtenir une équation de développante de cercle, il suffit de garder Z constant. Donc, pour obtenir sa tangente, il suffit de dériver par rapport à ϵ. Or, on a :

$$\frac{\partial X_1}{\partial \epsilon} = \frac{\partial X_1}{\partial \lambda_1} \cdot \frac{\partial \lambda_1}{\partial \epsilon}$$

Nos équations étant plus simples à dériver par rapport à λ_1, on obtiendra les tangentes à un coefficient près, $\frac{\partial \lambda_1}{\partial \epsilon} = cste$, mais cela n'a aucune importance puisque l'objectif est d'obtenir un pôle, qui est finalement l'intersection de deux droites tangentes aux points extrêmes.

Voici les calculs de dérivation par rapport à λ_1 :

$$\begin{cases} \frac{\partial X_1}{\partial \lambda_1}(\epsilon, Z) = Q'_{xz}(\lambda_1) \cdot Z + (Q'_{x\epsilon}(\lambda_1) \cdot \epsilon + Q_{x\epsilon}(\lambda_1) \cdot \epsilon') + Q'_x \\ \frac{\partial Y_1}{\partial \lambda_1}(\epsilon, Z) = Q'_{yz}(\lambda_1) \cdot Z + (Q'_{y\epsilon}(\lambda_1) \cdot \epsilon + Q_{y\epsilon}(\lambda_1) \cdot \epsilon') + Q'_y \end{cases}$$

avec :

$$\frac{\partial \epsilon}{\partial \lambda_1} = \frac{r_{b1} \cdot \epsilon_\gamma}{g_f + g_a + g_\beta} = cste$$

et :

$$\begin{cases} Q'_{xz} = -K_1 \cdot \sin\lambda_1 + K_2 \cdot \cos\lambda_1 \\ Q'_{x\epsilon} = -K_3 \cdot \sin\lambda_1 + K_4 \cdot \cos\lambda_1 \\ Q'x = -r'_1 \cdot \sin\lambda_1 - K_5 \cdot \sin\lambda_1 + K_6 \cdot \cos\lambda_1 \end{cases}$$

$$\begin{cases} Q'_{yz} = -K_2 \cdot \sin\lambda_1 - K_1 \cdot \cos\lambda_1 \\ Q_{y\epsilon} = -K_4 \cdot \sin\lambda_1 - K_3 \cdot \cos\lambda_1 \\ Q'y = -r'_1 \cdot \cos\lambda_1 - K_5 \cdot \cos\lambda_1 - K_6 \cdot \sin\lambda_1 \end{cases}$$

Le choix du point de passage

Notre choix de point passage a été guidé par l'existence d'une fonction SGDL permettant de calculer un point de passage de conique à partir d'un triangle de construction. Nous venions de déterminer un triangle de construction constitué de deux points de tangence et d'un pôle, nous avons donc décidé de nous passer du calcul d'un point de passage et d'utiliser cette fonction nommée SGy2_cir pour le déterminer.

La conique est une ellipse

Jusqu'à l'exécution du code, on ne connaissait pas *a priori* la forme de la conique générée, on avait seulement imposé quelques contraintes de passage et de tangence, le résultat graphique a donné une ellipse. En fait, l'affichage nous donne une surface elliptique lorsqu'on effectue une vue de face d'un cylindre elliptique.

Finalement, il s'avère que, lorsqu'on affiche des sphères sur les points de la courbe en développante de cercle pour la visualiser sur SGDLsoft, on s'aperçoit que cette approximation 2D a l'air visuellement très bonne.

L'obtention d'un profil symétrique

Dans le cadre de l'approche linéaire, les trapèzes étaient générés de manière symétrique ; par contre, dans le contexte d'une approximation quadratique, si l'ellipse approche bien une développante de cercle, elle n'approche aucunement le profil symétrique. Ainsi, pour obtenir les deux profils en développante d'une dent, il faut effectuer l'intersection entre deux ellipses calculées de manières identiques pour chaque profil en développante, comme le montre la figure 5.5.

5.4.2 Approximation de la dent par une quadrique

Introduction

Les quadriques sont l'extension volumique des coniques, donc pour construire une quadrique, il faut définir deux triangles de construction liés à la base par les points de tangence : on définit en fait 2 sections de coniques, dans des plans différents mais s'interséquant en des points de tangence, pour définir la quadrique. Ainsi, la première section a été déterminée dans le chapitre précédent, c'est une

ellipse, il reste donc à définir l'autre conique, c'est-à-dire le pôle de son triangle de construction et un point de passage.

Le choix du deuxième pôle

Comme pour l'approximation linéaire, le cas d'un engrenage droit est simple à traiter, puisque pour obtenir la continuité de matière il suffit d'envoyer le pôle à l'infini.

Dans le cas, d'un engrenage hélicoïdal par contre, il faut chercher à converger vers le profil suivant. Pour cela, nous avons déterminé un pôle comme l'intersection d'un plan et d'une droite : le plan est défini par les deux points de tangence du premier profil et un point de tangence du deuxième profil, la droite quant à elle est définie par les deux points de passage. C'est ce que montre la figure 5.6.

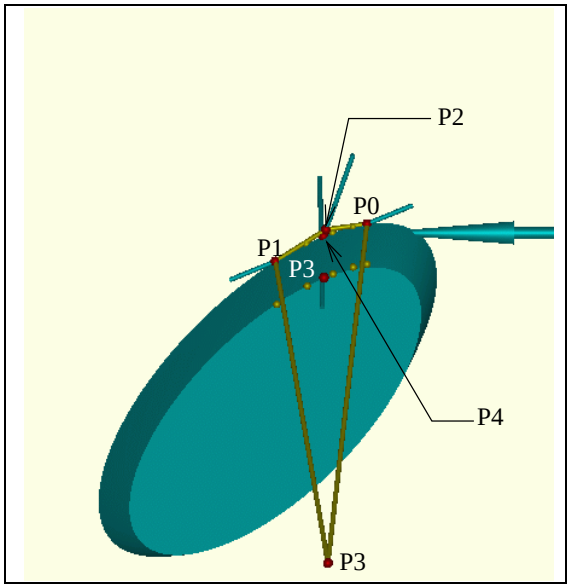


FIG. 5.6 – *Figure de construction de l'approximation quadratique de l'hélicoïde*

Le choix du deuxième point de passage

Nous avons choisi de placer le point de passage à peu près au milieu de la développante du profil suivant afin de répartir l'erreur de manière égale.

De plus, le fait de tenter de converger vers le deuxième profil montre qu'il y a une approximation : ainsi, avec 6 points de contrôle pour la quadrique, il n'est pas possible de passer précisément par le deuxième profil. Sur ce type d'approximation, nous avons atteint les limites des primitives à base de quadriques.

La quadrique est un hyperboloïde à deux nappes

Comme pour la détermination de la conique dans le plan, nous ne connaissons pas *a priori* la forme du volume obtenu en donnant les deux triangles de construction. Le calcul montre finalement

qu'on obtient une portion d'hyperboloïde à deux nappes à base elliptique.

Lorsqu'on coupe par deux plans parallèles cet hyperboloïde, pour ne garder que la partie utile, on constate que l'approximation du premier profil est évidemment très bonne, par contre en ce qui concerne le second profil, on constate déjà les futures discontinuités de matière au niveau visuel.

5.4.3 Les résultats de l'approche quadratique

Résultats qualitatifs

Les résultats obtenus sont nettement meilleurs par rapport à l'approche linéaire : en effet, en termes de nombre de quadriques, celui-ci a diminué très fortement, en obtenant en plus une approximation plus fidèle. Les résultats sont très bons, notamment pour les engrenages droits, puisqu'on modélise une dent complète à partir de seulement 5 quadriques, ces quadriques étant les suivantes:

- deux cylindres elliptiques infinis ;
- un volume compris entre deux plans infinis normaux à l'axe Z ;
- un cylindre infini de rayon r_{a1} (rayon de tête) ;
- un cylindre infini de rayon r_{A1} (rayon actif de pied).

Dans le cas des engrenages hélicoïdaux, ce modèle à 5 quadriques est le même, mais il est utilisé pour générer une dent élémentaire, la dent complète est alors obtenu par empilage de ces dents élémentaires. Enfin, pour obtenir une meilleure continuité des raccords, les cylindres elliptiques sont déformés de manière continue en hyperboloïdes à deux nappes. La figure 5.7 illustre le type de dents que nous sommes capables de produire avec cette approche.

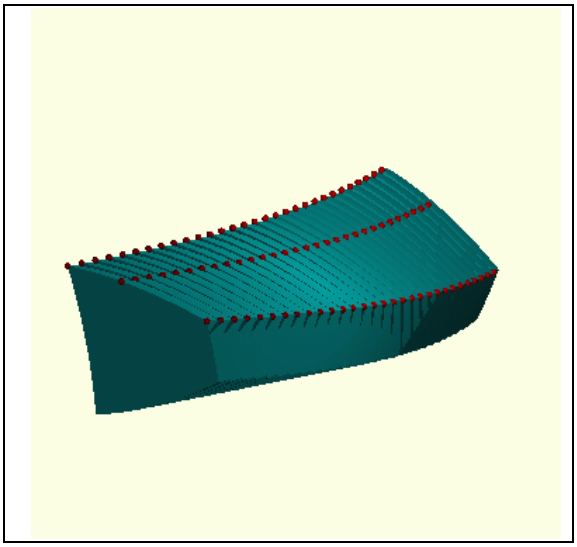


FIG. 5.7 – *Résultat final de l'approximation quadratique de la denture hélicoïdale*

Résultats numériques

Dans tout cette approche, il était devenu essentiel de chiffrer une erreur de manières analytique et numérique. Dans cette optique, un calcul numérique a pu rapidement aboutir en effectuant l'intersection de la normale au point I sur la développante de cercle (au niveau du rayon primitif) avec l'ellipse obtenue sous SGDLsoft. Voici les conditions du calcul (les caractéristiques de la dent autrement dit) :

- $m_0 = 5\ mm$;
- $\alpha_0 = 20^o$;
- $\beta_0 = 0^o$;
- $b = 50\ mm$;
- $x_1 = 0\ mm$.

On obtient ainsi une erreur approximative d'une valeur $\delta = 0.021\ mm$. Ce résultat confirme l'appréciation visuelle très correcte observée sur les différentes images calculées.

5.4.4 Améliorations et conclusions

L'approximation de la développante de cercle pourra être encore améliorée en choisissant un point de passage non pas calculé par la fonction SGy2_cir, mais en calculant directement un point de la développante, par exemple le point I. Et puis surtout, en utilisant des primitives géométriques de degré supérieur, on obtiendra très certainement la continuité de matière et des tangentes, mais nous y reviendrons au chapitre 7.

Pour conclure sur cette approche mathématique de la représentation des engrenages, nous avons essayé dans un premier temps de modéliser de manière conventionnelle la denture de l'engrenage à l'aide de blocs trapézoïdaux, mais cette approximation est très lourde à mener en termes de calculs, c'est pourquoi nous avons envisager dans un deuxième temps une modélisation plus en adéquation avec les potentialités du logiciel SGDLsoft et nous avons tenté d'approximer l'hélicoïde de manière quadratique. Nous avons pu ainsi obtenir une très bonne approximation du profil en développante tout en essayant de suivre l'hélice de la denture.

Toutefois, nous n'avons pu éviter des discontinuités de matière en suivant cette hélice, aussi bien en approximation linéaire qu'en approximation quadratique, mais l'utilisation des quadriques nous a tout de même permis d'alléger très fortement l'arbre volumique d'une telle modélisation en nous rapprochant encore plus de l'hélice et nous avons finalement atteint les limites de l'utilisation de telles primitives.

Chapitre 6

L'approche informatique

6.1 Introduction

Au long de notre projet, nous avons eu tout d'abord une approche véritablement mécanicienne en modélisant le processus d'obtention des engrenages. Ensuite, nous avons abordé la problématique des engrenages de manière vraiment mathématique, avec la recherche d'équations générales. Enfin, nous avons aussi étudié cette problématique dans le contexte informatique et infographique.

Les problèmes en la matière sont évidemment nombreux et nous ne nous attarderons que sur les plus importants, notamment dans l'amélioration des temps de calcul, mais aussi dans les erreurs provenant de la précision machine des calculs.

6.2 L'optimisation du code

6.2.1 Les algorithmes

Scheme et SGDL sont intimement liés par le λ-calcul on l'a déjà vue au chapitre 3 ; en conséquence, toute tentative de conception volumique doit répondre à une logique de construction.

De plus, nous avons cherché tout au long de notre projet à écrire des algorithmes simples, rapides et sans cas particulier, notamment en bannissant toute programmation impérative ou utilisant de nombreux tests logiques. Cette méthode nous a conduit à obtenir un code très concis : par exemple, pour l'approche fabrication, le corps du programme est une boucle récursive qui tient en à peine 10 lignes (*cf.* annexe F), le reste ne consiste qu'en la définition de volumes de base ou de fonctions géométriques.

6.2.2 La programmation fonctionnelle

Intérêt

Tous nos programmes sont constitués de fonctions réutilisables à loisir. L'intérêt pour nous, c'est de pouvoir définir des scènes volumiques complexes en faisant appel à des fonctions élémentaires dépendant uniquement de quelques paramètres. Par exemple, l'obtention d'une scène composée de

deux engrenages de taille différente fait appel à la même fonction `engrenage`, ce qui change ce sont les paramètres donnés en entrée et rien d'autre.

L'utilisation des fonctions projectives

Au début de la définition de notre outil crémaillère, nous avons décidé de le paramétrer de manière projective. Cela nous a permis d'avoir une définition unique pour la cinématique entre la roue et la crémaillère : par exemple, si l'on voulait obtenir un engrenage hélicoïdal, il suffisait de déformer la crémaillère en lui inclinant sa denture simplement en jouant sur un paramètre, la base du prisme au lieu d'être rectangulaire devenait alors un parallélogramme.

Par contre, lorsqu'on a voulu ajouter un arrondi en tête de dent, les calculs analytiques se sont avérés très complexes (*cf.* annexe E), mais ils ont abouti, même s'ils ont conduits à des expressions beaucoup trop lourdes au niveau trigonométrique (pouvant conduire rapidement à des imprécisions numériques difficiles à contrôler...). Par la suite, avec une meilleure connaissance des outils projectifs de SGDLsoft, tous ces calculs ont disparu au profit de l'application de transformations élémentaires (translations, projections...) de points connus (*cf.* annexe F).

Bibliothèques mathématiques

– *La fonction involute :*

La fonction involute (*cf.* paragraphe 5.2.1) est souvent utilisée en matière d'engrenages, elle est en effet utile pour définir l'équation de la développante de cercle, ainsi que pour calculer des angles importants (comme α'_t) après l'application d'un déport de denture. L'équation de l'involute est : $inv\alpha = \tan\alpha - \alpha$, d'où sa fonction dérivée $inv'\alpha = \tan^2\alpha$.

En fait, on a souvent le problème de connaître α sachant que $inv\alpha = K = cste$. Tout le problème revient donc à trouver la fonction réciproque de l'involute. Pour cela, l'étude de la fonction montre que l'involute est strictement croissante sur $]-\frac{\pi}{2},\frac{\pi}{2}[$ et aussi bijective de $]-\frac{\pi}{2},\frac{\pi}{2}[$ sur $\Re$. Par conséquent, l'équation $inv\ \alpha - K = 0$ admet toujours une solution $\alpha_{solution} \in]-\frac{\pi}{2},\frac{\pi}{2}[$, comme le montre la figure 6.1. Enfin, pour ce qui est du calcul de la fonction réciproque, nous avons utilisé l'algorithme de Newton-Raphson.

– *L'algorithme de Newton-Raphson :*

Pour résoudre une équation une équation du type $f(x) = 0$, l'utilisation de l'algorithme de Newton-Raphson est intéressant car il a la propriété de converger très rapidement vers la solution. Par contre, pour éviter les risques de divergence, il faut donner une valeur initiale proche de la solution. L'expression de cet algorithme est la suivante :

$$X_{n+1} = X_n - \frac{f(X_n)}{f'(X_n)}$$

On arrête quand $\mid X_{n+1} - X_n \mid < \epsilon$.

Il est à noter que pour notre algorithme (*cf.* annexe H), pour initialiser l'algorithme, on utilise un développement limité d'ordre 1 de la fonction involute.

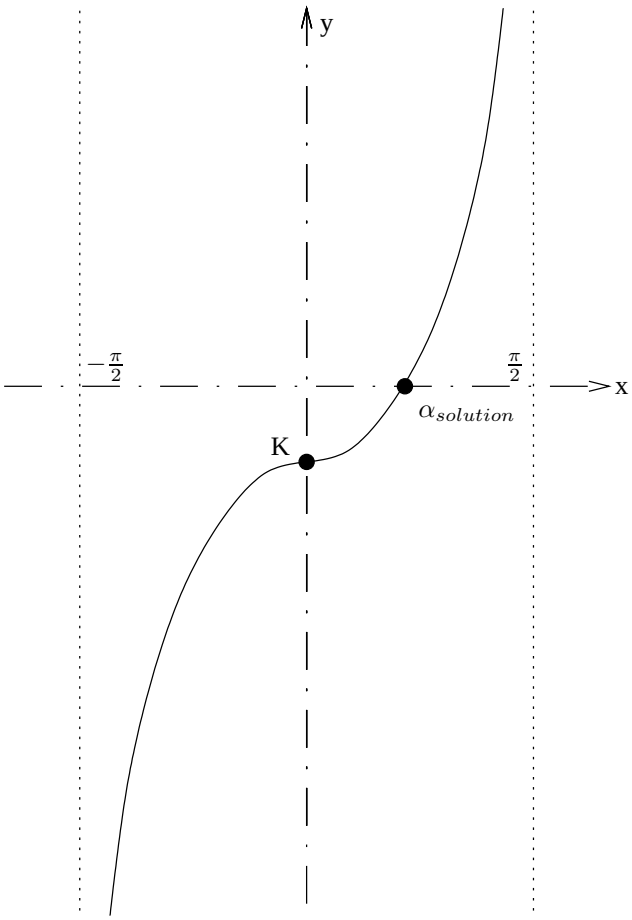


FIG. 6.1 – *Courbe représentative de la fonction* $f(x) = inv\ x - K$

– *La fonction intersection roue - crémaillère :*

Cette fonction permet de déterminer le point P (*cf.* annexe 6.2) nécessaire à le détermination de la quadrique englobante optimale pour la crémaillère de taillage : l'ordinateur ne calculera des intersections de celle-ci avec la roue que dans cette zone, la zone utile de taillage (*cf.* annexe F). Le problème dans le plan revient donc à calculer l'intersection d'un cercle avec une droite, comme le montre la figure 6.2.

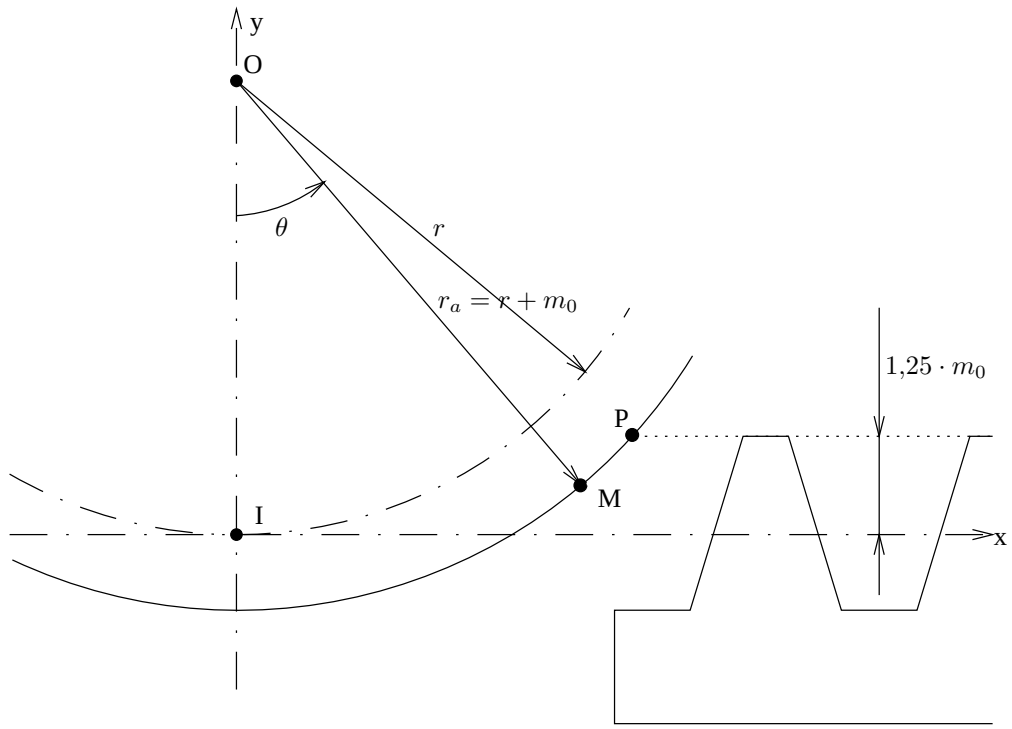


FIG. 6.2 – *Figure de l'intersection roue - crémaillère*

D'après les positions relatives de la crémaillère, les équations sont les suivantes :

- équation du cercle : $x^2 + y^2 = r_a^2 = (r + m_0)^2$
- équation de la droite : $y = -r + 1{,}25 \cdot m_0$
- point d'intersection : $P(x_P,y_P)$ ou (r_a,θ_P), avec :

$$\begin{cases} x = (r + m_0) \cdot \sin\theta \\ y = -(r + m_0) \cdot \cos\theta \end{cases}$$

D'où les résultats suivants :

$$\begin{cases} x_P = \pm\sqrt{4{,}5 \cdot m_0 \cdot -\frac{1}{16} \cdot m_0^2} \\ \theta_P = \pm \arcsin(-\frac{1}{r+m_0} \cdot \sqrt{4{,}5 \cdot m_0 \cdot -\frac{1}{16} \cdot m_0^2}) \end{cases}$$

- *Fonctions annexes (cf. annexe H) :*
  - Conversion degrés $\leftrightarrow$ radians ;
  - Fonction tracé de repère ;
  - Fonction de visualisation de plan.

6.3 L'optimisation des temps de calculs

6.3.1 Introduction aux quadriques englobantes

La problématique des quadriques englobantes, aussi appelées min-max, est en général complètement étrangère à un utilisateur de CAO : il s'agit en fait d'une technique infographique permettant d'accélérer le traitement du rendu en plaçant au plus haut dans l'arbre volumique une quadrique qui englobe l'objet à représenter, cela permet d'optimiser les tests d'exclusions (*cf.* paragraphe 3.1.5).

Pour gagner du temps de traitement en plaçant un min-max autour d'un objet complexe, il faut que cette forme soit fermée, et la seule quadrique qui ait cette propriété, c'est l'ellipsoïde. D'une manière générale, les min-max sont donc des ellipsoïdes, reste le calcul de leurs points de contrôle. Pour travailler rapidement, le plus simple est de définir un hexaèdre projectif, dans lequel est inscrit l'objet complexe, et puis à l'aide de fonction SGDLsoft, il est possible de calculer la quadrique circonscrite à cet hexaèdre.

6.3.2 L'approche taillage

Les quadriques englobantes

Dans notre code, nous n'avons pas utilisé de fonction SGDLsoft prédéfinies, nous sommes remontés jusqu'aux 6 points de contrôle de la quadrique, paramètres d'entrée de la fonction de référence DLformz2. Par conséquent, nos engrenages ne comportaient aucun min-max.

Afin d'améliorer les temps de calcul et de traitement du rendu volumique, nous avons placé des boîtes englobantes, et la difficulté est de bien les choisir. Lorsqu'on taille une roue, ce qui est important de contrôler c'est la matière enlevée, autrement dit, il faut mettre des min-max sur les dents qui taillent uniquement et à un instant précis. Pour cela, nous avons développé une fonction qui calcule l'intersection entre la crémaillère et la roue (se reporter au paragraphe 6.2.2) et nous avons placé une boite englobant uniquement les dents utiles au taillage.

L'allégement de l'arbre volumique

Lorsqu'on taille la roue, on utilise une crémaillère d'une longueur égale à la circonférence du cercle primitif. Par contre, pour éviter que l'arbre volumique ne devienne trop complexe, nous avons fait en sorte de ne recalculer que la partie de crémaillère utile à l'instant donné. Pour cela, nous avons appliqué des règles de proportionnalité permettant de déterminer la dent utile à chaque incrément de taillage (*cf.* annexe F).

6.3.3 L'approche mathématique

L'approximation linéaire

Cette approche a consisté, rappelons-le, en un empilage de boîtes à base trapézoïdale. Nous avons donc placé des min-max à chaque niveau de cette arborescence : les calculs des boîtes sont élémentaires pour les boites à base trapézoïdale et pour les dents élémentaires (*cf.* annexe F et figure 6.3).

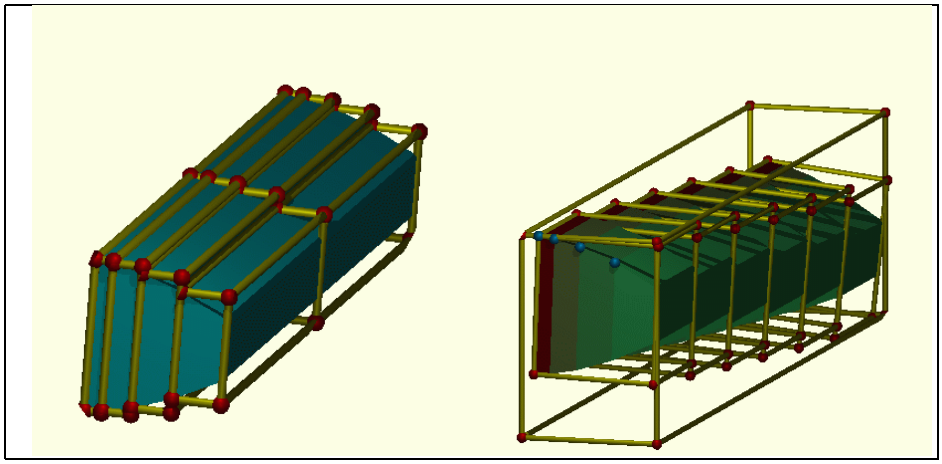


FIG. 6.3 – *Hexaèdres de construction des quadriques englobantes de l'approximation linéaire*

Par contre, la détermination d'un min-max englobant une dent complète a été plus délicat, il a nécessité l'écriture d'une fonction calculant les points minima et maxima en X, Y et Z, cette fonction travaillant dans le cas général d'une dent quelconque, la boîte obtenue est assez grande et pourrait être affinée.

L'approximation quadratique

L'arbre volumique de génération d'une dent complète pour l'approximation linéaire est le suivant :

– dent complète ;

– dent élémentaire.

Les min-max utilisé sont les mêmes que précédemment (*cf.* annexe G et figure 6.4).

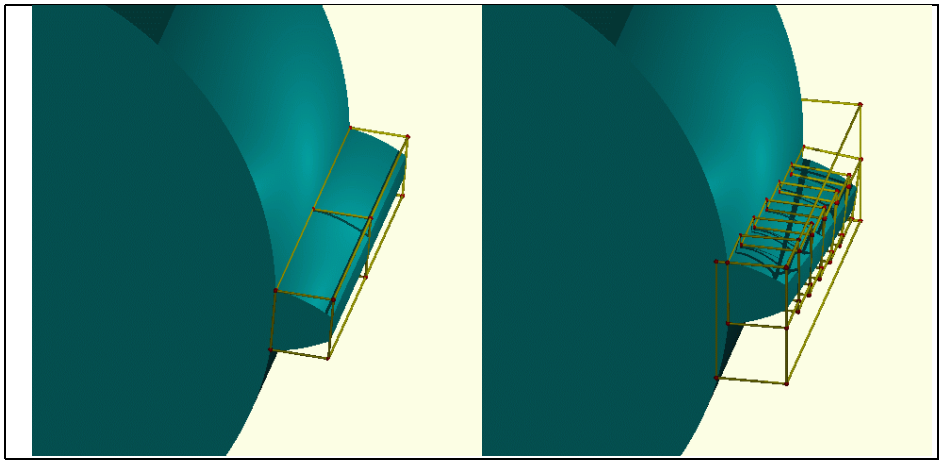


FIG. 6.4 – *Hexaèdres de construction des quadriques englobantes de l'approximation quadratique*

Chapitre 7

Applications

7.1 Les images

Le format de sortie standard sur SGDLsoft est le format targa (*.*tga*). Ce format a l'intérêt d'être simple à coder et d'être codé sur 24 bits, ce qui permet d'obtenir des images de très bonne qualité.

Par ailleurs, grâce à une programmation entièrement fonctionnelle et paramétrable, il est possible d'obtenir un grand nombre de scènes volumiques complexes et variées. Entre autres, nous avons appairé très facilement deux engrenages hélicoïdaux à axes perpendiculaires,comme le montre la figure 7.1 ; nous avons ensuite effectué des coupes pour montrer la matière ainsi que la zone de contact, c'est ce que montre la figure 7.2.

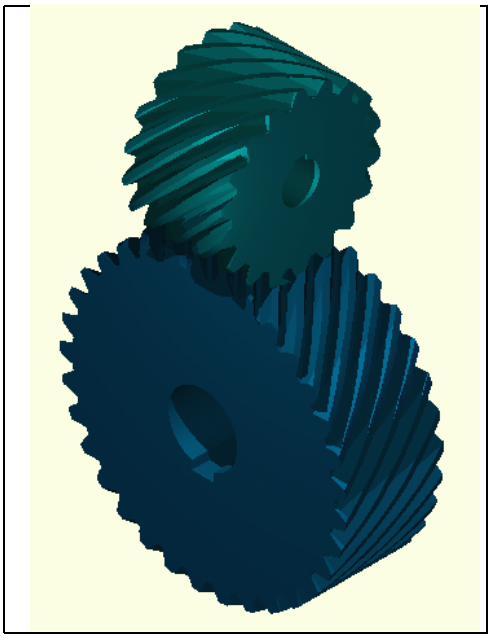


FIG. 7.1 – *Paire d'engrenages hélicoïdaux à axes perpendiculaires*

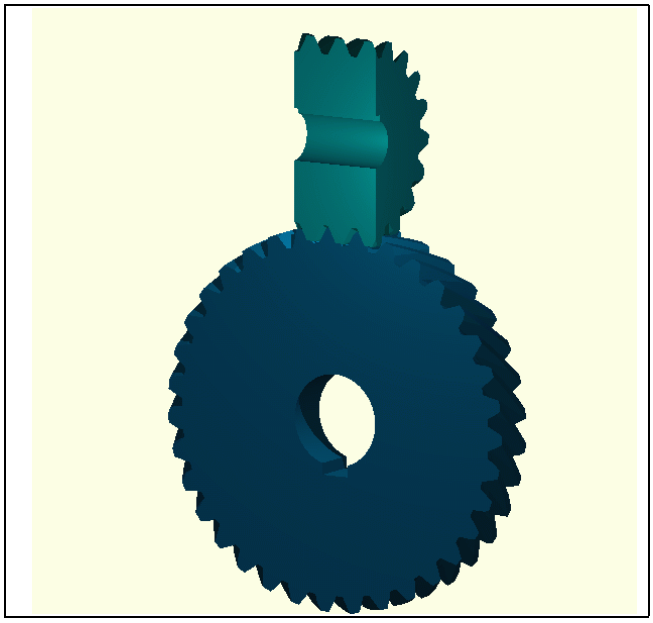


FIG. 7.2 – *Paire d'engrenages hélicoïdaux à axes perpendiculaires en coupe*

Enfin, il est possible, grâce au codage ternaire des densités, de filtrer la matière. L'intérêt réside, par exemple, dans la visualisation de la surface d'un volume sans la matière intérieure. Dans notre cas, sur la figure 7.3, nous avons filtré la "peau" d'un engrenage, cela nous a permis de mieux identifier le profil étudié.

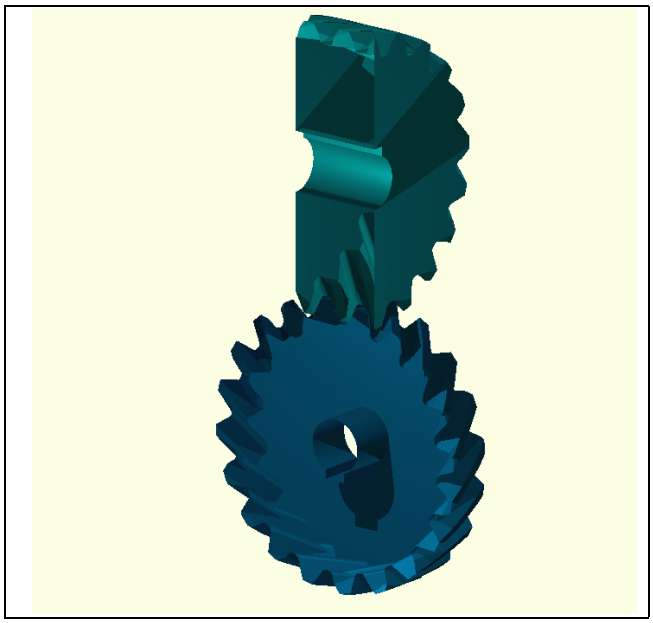


FIG. 7.3 – *Vue de la "peau" d'un engrenage*

7.2 L'étude du contact

En matière d'engrenage, lorsque l'on commence à travailler sur des profils un peu particulier, la détermination de la surface de contact est très délicate.

Pour tester la validité de nos travaux en matière de contact dans l'objectif de simulation ou de prévision du contact, nous avons dû en premier lieu optimiser le code : en effet, pour tendre vers une solution théorique, il faut avoir une approximation assez précise. Or, pour notre projet, nous avons mené en parallèle des études à caractères très divers, et comme le travail d'optimisation n'a pas été du tout négligeable, nous n'avons pas eu le temps de réellement approfondir l'étude du contact.

Cependant, nous avons quand même cherché à retrouver les résultats classiques de mécanique, avec la recherche de la ligne de contact entre deux engrenages droits, mais aussi la ligne de contact entre deux engrenages hélicoïdaux à axes parallèles. On obtient des images satisfaisantes, (*cf.* figure 7.4) montrant les surfaces de contacts : ces surfaces sont planes, ce qui est dû au mode de génération de l'image (on a utilisé l'algorithme de l'approche "taillage" : c'est la partie plane des flancs plans de la crémaillère qui génère le profil actif de la dent).

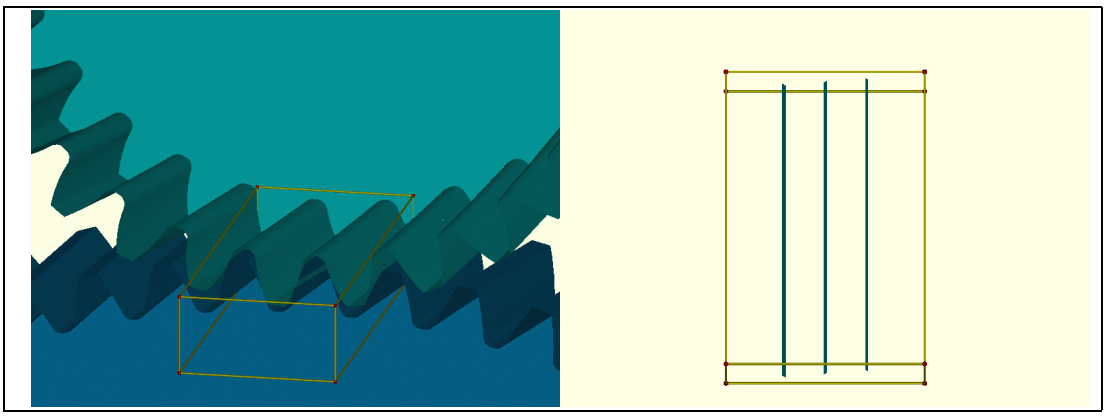


FIG. 7.4 – *Vue du contact au niveau d'une dent*

Par contre, pour juger à sa juste valeur l'intérêt de nos premiers résultats, il faut savoir ce que l'on cherche à déterminer : par exemple, s'il s'agit de connaître la nature géométrique du contact, cette approche convient parfaitement ; par contre, si l'objectif est de comparer rapidement le modèle mathématique et l'approximation informatique, il convient dans ce cas de pousser la recherche à partir de nos travaux sur l'approche mathématique.

7.3 Les animations

La création d'animations à partir de SGDLsoft n'est pas un problème : en effet, il suffit de stocker suffisamment d'images pour obtenir la fluidité désirée. Ensuite, il faut utiliser un logiciel permettant de générer des animations : dans notre cas, il s'agit de QuickTime Pro 4.0, le format de sortie des animations est donc **.mov*.

En ce qui concerne la préparation de l'animation, comme on travaille toujours en paramétrant la scène volumique avec un code Scheme, il est très facile d'effectuer des déplacements de l'observateur ou de modéliser les mouvements relatifs des objets composant la scène. Dans cette optique, nous avons généré des animations de trois types :

– tout d'abord, nous avons montré le phénomène de génération d'un engrenage par taillage à l'aide d'un outil crémaillère : nous avons donc calculé deux animations, la première montrant le taillage d'un engrenage droit, la seconde montre le taillage d'un engrenage hélicoïdal ; ces animations n'ont pas la prétention de simuler un mode d'usinage réel, elles ont simplement pour but de montrer comment on peut obtenir des engrenages par enlèvement de matière.

– ensuite, nous avons calculé une animation mettant en évidence une loi cinématique entre deux engrenages : en effet, les mouvements des deux engrenages étant liés par un rapport de réduction, on peut constater visuellement les différences relatives des vitesses de rotation.

– enfin, nous avons monté une animation illustrant le caractère projectif de la définition de la dent : cette animation montre comment la variation du paramètre β_0 influe sur la forme des dents de la crémaillère et, par conséquent, sur celles de l'engrenage généré. En outre, on peut observer la dépendance du diamètre d'un engrenage par rapport à l'angle d'hélice β_0.

7.4 Les passerelles avec d'autres formats

7.4.1 L'import de fichiers

Actuellement, il existe des convertisseurs permettant de solidifier des fichiers aux formats DXF (AutoCAD…) et VRML : des études effectuées au sein du GRCAO ont en effet mis en évidence l'intérêt de disposer de modèles véritablement volumiques (topologie des surfaces complètement intégrée…), notamment en matière d'architecture et d'aménagement urbain.

7.4.2 L'export vers POV-Ray

Après avoir calculé une scène volumique sous SGDLsoft, il est possible de la convertir au format POV-Ray. L'intérêt réside dans la possibilité d'appliquer alors des textures et ainsi d'obtenir des scènes volumiques avec un rendu photo-réaliste très poussé.

Nous avons fait un essai d'export, en choisissant pour thématique le moulage d'un engrenage, nous avons donc créé une scène volumique comportant les principaux éléments relatifs au moulage, ainsi on pouvait retrouver le modèle en bois, le noyau en résine, le moule en sable avec empreinte et enfin l'engrenage moulé en fonte. Le résultat n'est pas du plus bel effet, mais nous avons manqué de temps pour trouver des textures plus adéquates (*cf.* figure 7.5).

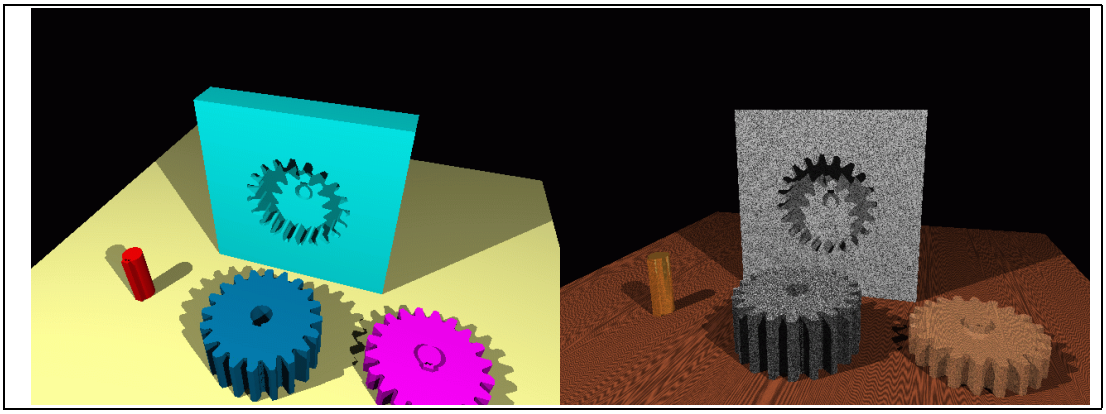


FIG. 7.5 – *Etude de texturing sur une scène volumique*

7.4.3 La voxellisation

Un autre format d'export intéressant est celui de la voxellisation : en effet, le découpage en petits cubes (des voxels, ou pixels en 3D) d'un volume permet de passer sur des logiciels du domaine de l'imagerie médicale très performants. Il est alors possible de faire toute une gamme de coupe ou de filtre de matière, avec en plus la possibilité de donner des effets de transparence.

7.5 La prochaine version de SGDLsoft

La compagnie SGDL systèmes Inc. finalise actuellement la prochaine version de SGDLsoft, dans la perspective d'une commercialisation courant octobre 1999. Cette version offrira comme principale caractéristique la possibilité de représenter des quartiques (*cf.* figure 7.6).

Il s'agit encore une fois d'une grande avancée, car ces surfaces définissant des volumes n'ont jusqu'à l'heure actuelle été que très peu étudiées. En effet, si l'on part de l'équation générale implicite sous forme polynômiale, il est quasiment impossible de donner un sens aux coefficients et par la même occasion de générer un volume ayant certaines propriétés géométriques. Toute la force de la nouvelle version réside donc dans l'initialisation de ces coefficients à partir de points et de contraintes.

Enfin, un navigateur programmé en java à été développé afin de permettre à terme une navigation volumique en temps réel.

Ce sont des avancées considérables dans le monde de la CAO, où la quasi totalité des moteurs 3D approchent les volumes par des polygones ou des polyèdres auxquels ils appliquent des algorithmes de rendu pour adoucir la surface.

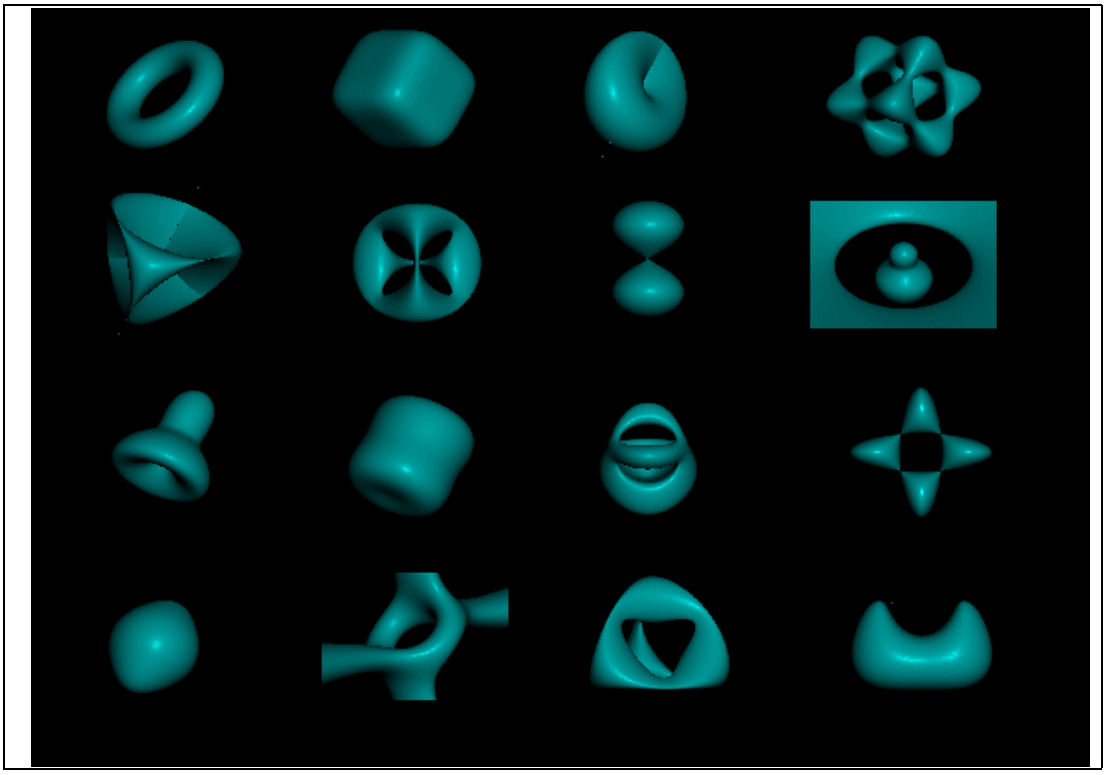


FIG. 7.6 – *Les quartiques et leurs possibilités*

Chapitre 8

Conclusions

8.1 Le bilan des résultats

8.1.1 Des objectifs globalement atteints

Pour nous, les principaux objectifs de notre projet ont été atteints :

- nous avions pour première étape la modélisation de deux engrenages hélicoïdaux à axes perpendiculaires, nous avons obtenu bien plus, car nos choix de modélisation sont restés généraux, nous pouvons donc modéliser une large partie de la famille des engrenages : en fait, notre modélisation se veut poussée et rapide.

- ensuite nous avions aussi pour objectif d'étudier les possibilités de fournir des résultats sous forme pédagogique ; dans ce cadre, les animations se sont révélées être un très bon vecteur de communication et de vulgarisation de phénomènes complexes.

Par contre, le temps nous a manqué pour finalisé l'étude du contact. En effet, avant de pouvoir retrouver des résultats théoriques, il faut que le code soit bien optimisé. Enfin, nous n'avons pas terminé le chiffrage de l'erreur : si nous pouvons obtenir une valeur numérique, il reste à développer des algorithmes donnant l'écart maximum entre le modèle mathématique et le modèle informatique, et puis il serait aussi intéressant d'effectuer le calcul en analytique à partir des équations mathématiques et des équations des quadriques.

8.1.2 Un succès en matière de veille technologique

Nous ne connaissions pas du tout les potentialités de SGDLsoft avant d'arriver à Montréal, même si les premiers contacts que nous avions eu avec M. Rotgé laissaient présager de bonnes perspectives de travail. Il était par conséquent loin d'être évident que la problématique des engrenages puissent s'appliquer au modeleur volumique SGDLsoft.

D'ailleurs, nos travaux ont suivi les potentialités offertes par le logiciel, notamment en matière de géométrie projective et d'algorithmique, ce qui a eu pour conséquence de rendre notre étude plutôt

originale en matière de modélisation d'engrenages.

Enfin, il faut rappeler que l'approximation d'une hélice de manière quadratique n'a quasiment jamais été traitée : en termes de recherche pure, nous apportons donc une nouvelle approche dans l'approximation des surfaces hélicoïdales.

8.2 Les évolutions futures

Le grand intérêt de notre travail réside aussi dans les opportunités et perspectives qu'il offre en matière d'engrenages. Ainsi, suite à nos travaux, plusieurs types d'approche pourraient être envisageables :

– *une approche à base de quartiques :*

  L'étude pouvant venir immédiatement dans la continuité de notre projet serait de traiter l'approche mathématique avec une approximation à base de quartiques, cela permettrait de réduire le nombre de volumes élémentaires nécessaires pour approcher une dent complète, et le grand nombre de contraintes à définir permettra certainement d'obtenir la continuité de matière et des tangentes au niveau des raccords.

– *la modélisation du pied de dent :*

  Nous avons aussi travaillé sur la possibilité d'ajouter un pied de dent à notre modèle mathématique ; *a priori*, il n'y a aucun problème, mais ce travail n'a pas pu aboutir car les premiers résultats semblaient contenir des erreurs, en conséquence le manque de temps nous a contraint à laisser cette évolution de côté.

– *la voxellisation ou la polygonisation :*

  Les développeurs de SGDLsoft étudient actuellement le développement de passerelles vers le monde des polygones, cette volonté répond à une attente des utilisateurs, il peut s'avérer intéressant de pouvoir reprendre des modèles volumiques sous les logiciels plus conventionnels de CAO. Enfin, la voxellisation ou la polygonisation pourrait permettre éventuellement de retrouver des sorties éléments finis.

– *le développement d'outil de visualisation :*

  Il existe en effet un grand nombre de phénomènes mécaniques sur les engrenages qui sont délicats à expliquer, le développement d'outil pédagogique peut donc se révéler être intéressant dans cette optique, on citera l'exemple des interférences de taillage ou de fonctionnement, ainsi que les problèmes relatifs au contact, il serait en effet intéressant de montrer, par des animations, la génération de lignes d'engrènement.

– *l'application à la commande numérique :*

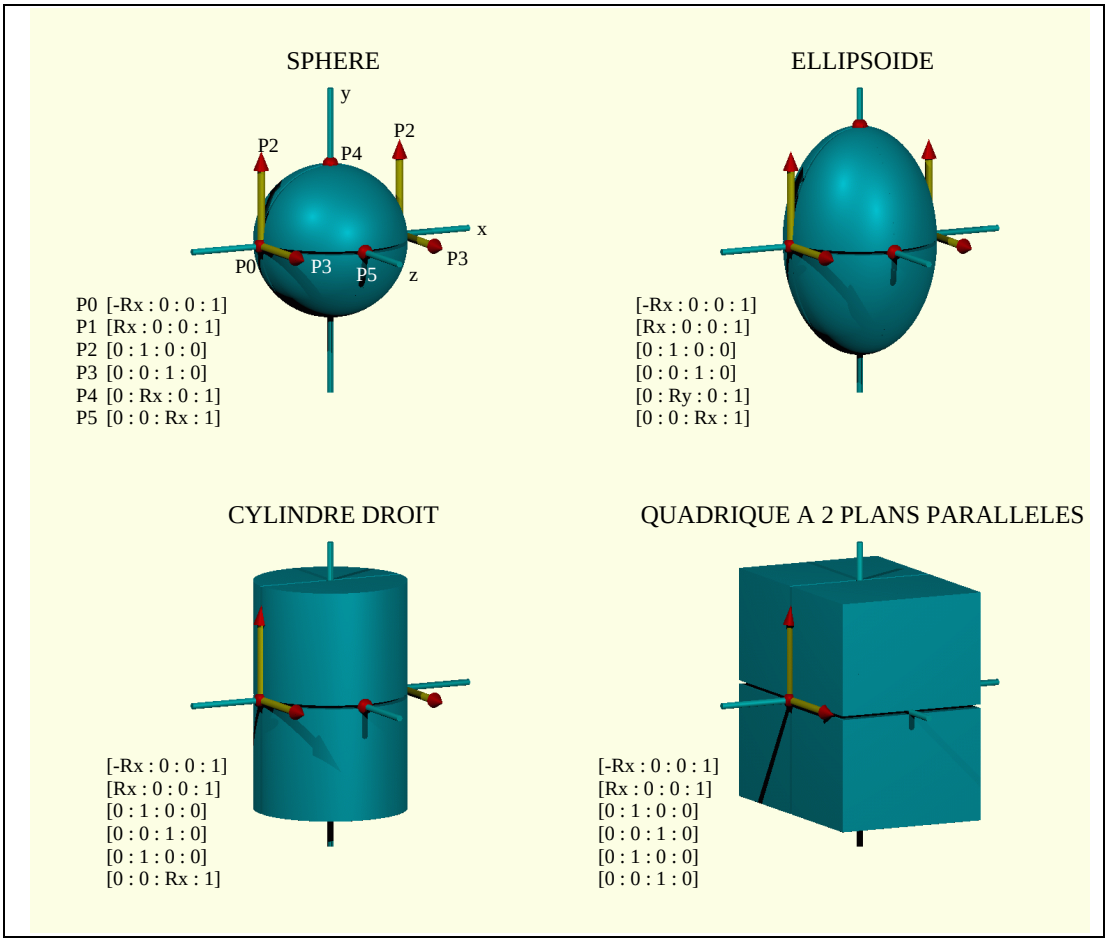
  En matière de taillage des engrenages, les machines à commandes numériques 5 axes sont en passe de devenir un nouveau procédé d'obtention, pour cela il faudra être capable de donner des trajectoires d'outils précises et relatives aux surfaces modélisées. Enfin, une autre application directe sera alors la simulation d'usinage (CFAO), car connaissant l'objet à réaliser et la trajectoire de l'outil de coupe et sa géométrie, dans un environnement programmable tel que celui de SGDLsoft, il sera possible de modéliser l'usinage d'un engrenage sur une machine à commande numérique.

Deuxième partie

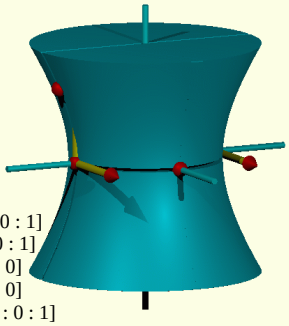
Annexes

Annexe A

Différentes quadriques et leurs points de contrôle

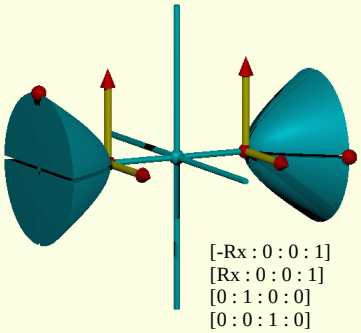


HYPERBOLOIDE A UNE NAPPE



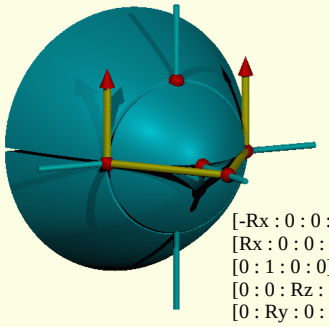
[-Rx : 0 : 0 : 1]
[Rx : 0 : 0 : 1]
[0 : 1 : 0 : 0]
[0 : 0 : 1 : 0]
[-Rx : Ry : 0 : 1]
[0 : 0 : Rx : 1]

HYPERBOLOIDE A 2 NAPPES



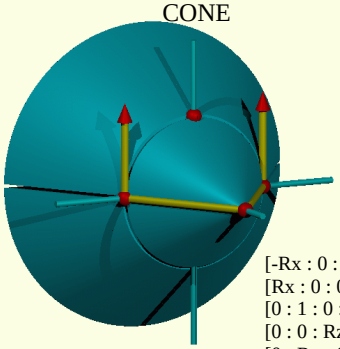
[-Rx : 0 : 0 : 1]
[Rx : 0 : 0 : 1]
[0 : 1 : 0 : 0]
[0 : 0 : 1 : 0]
[-(Rx+dx) : Ry : 0 : 1]
[Rx+dx : 0 : Rz : 1]

PARABOLOIDE ELLIPTIQUE



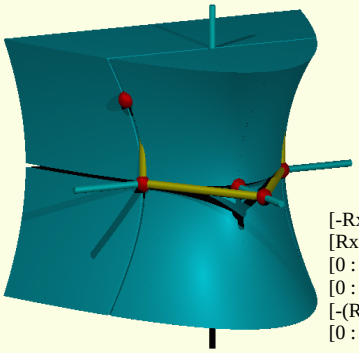
[-Rx : 0 : 0 : 1]
[Rx : 0 : 0 : 1]
[0 : 1 : 0 : 0]
[0 : 0 : Rz : 1]
[0 : Ry : 0 : 1]
[0 : 0 : Rz : 2]

CONE



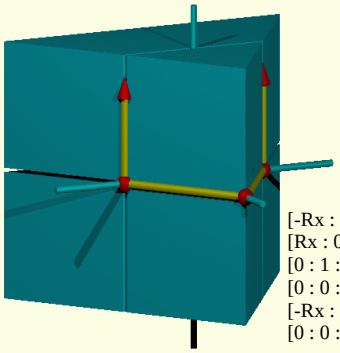
[-Rx : 0 : 0 : 1]
[Rx : 0 : 0 : 1]
[0 : 1 : 0 : 0]
[0 : 0 : Rz : 1]
[0 : Ry : 0 : 1]
[0 : 0 : Rz : 1]

PARABOLOIDE HYPERBOLIQUE



[-Rx : 0 : 0 : 1]
[Rx : 0 : 0 : 1]
[0 : 1 : 0 : 0]
[0 : 0 : Rz : 1]
[-(Rx+dx) : Ry : 0 : 1]
[0 : 0 : Rz : 2]

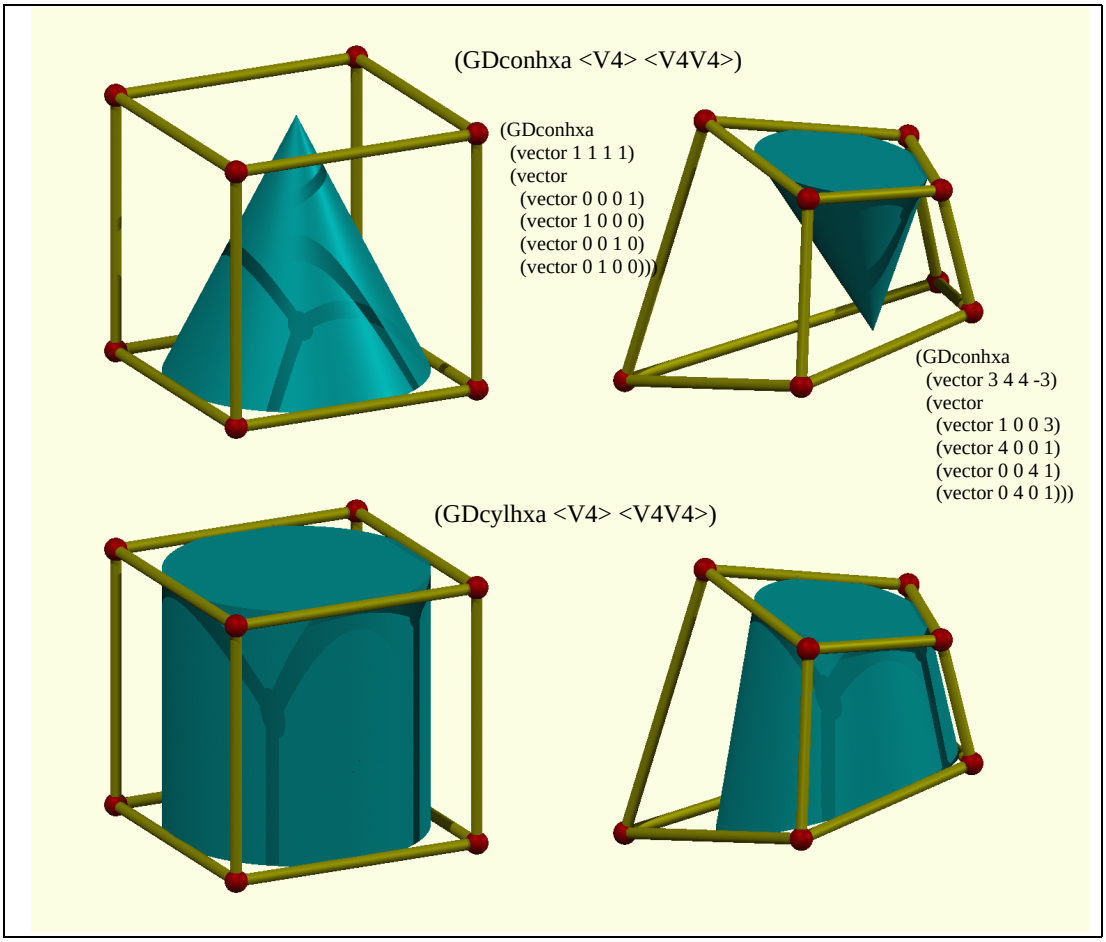
ANGLE DIEDRE SOLIDE



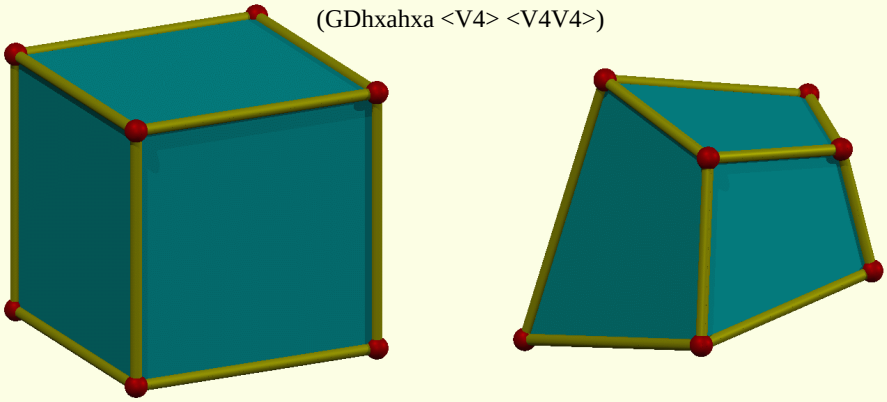
[-Rx : 0 : 0 : 1]
[Rx : 0 : 0 : 1]
[0 : 1 : 0 : 0]
[0 : 0 : 1 : 0]
[-Rx : Ry : 0 : 1]
[0 : 0 : Rz : 1]

Annexe B

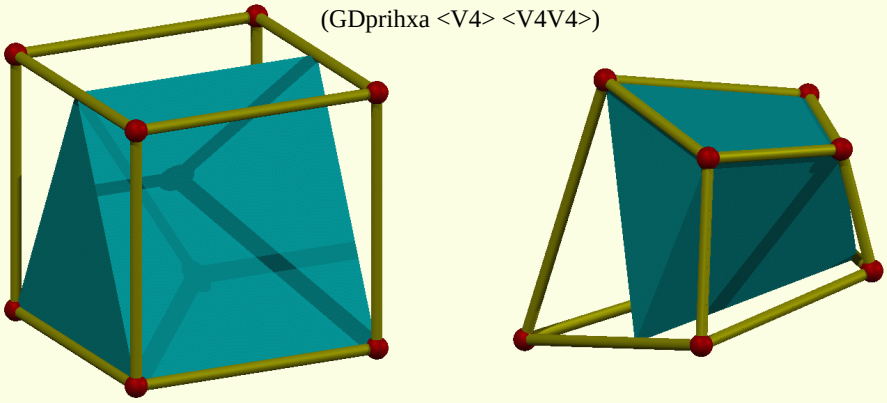
Les principales primitives de construction précontraintes



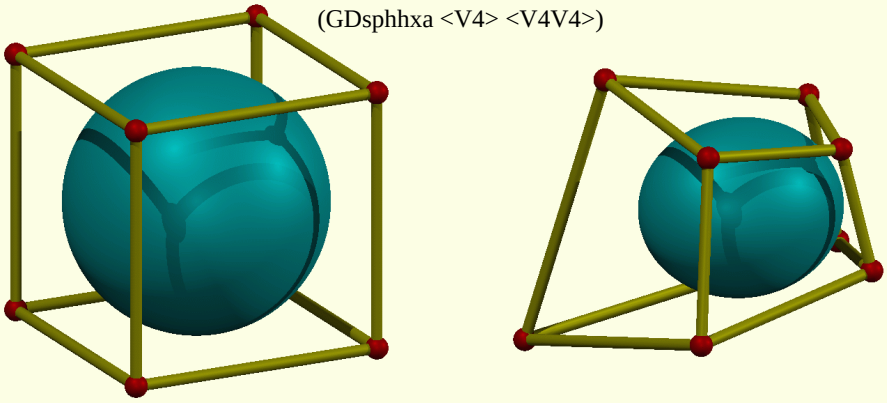
(GDhxahxa <V4> <V4V4>)



(GDprihxa <V4> <V4V4>)



(GDsphhxa <V4> <V4V4>)



Annexe C

Les fonctions SGDL les plus utilisées pour notre projet

Hormis les primitives définies dans l'annexe B, le code que nous avons développé utilise essentiellement les fonctions qui suivent. Pour les autres, se reporter au Dictionnaire Géométrique SGDL [15].

(DLatt <attribut>...<*volume*>)	Cette fonction permet d'associer des attributs (couleur, matrice de changement de repère...) à un objet volumique dont la géométrie est spécifiée par le dernier argument <*volume*>.
(DLdif <*volume*><*volume*>...)	Cette fonction retourne le volume <*volume*> issu de la différence positive de n volumes <*volume*><*volume*>....
(DLdua <*volume*>)	Cette fonction retourne le volume <*volume*> issu du dual ternaire du volume <*volume*>(on obtient en fait le volume complémentaire du volume <*volume*>, par inversion des densitś).
(DLformz2 <V6V4>)	Cette fonction retourne le volume <*volume*> issu de la quadrique générale définie par les 6 points de contrôle du vecteur <V6V4>.
(DLint <*volume*><*volume*>...)	Cette fonction retourne le volume <*volume*> issu de l'intersection de Kleene de n volumes <*volume*><*volume*>....
(DLmmx <*quadrique*><*volume*>)	Cette fonction permet d'attribuer à un volume <*volume*> une quadrique englobante définie par vous-même, afin d'accélérer les post-traitements, comme le traitement graphique. C'est pourquoi toutes les fonctions de la sous-librairie GD englobent automatiquement les primitives géométriques générées. Il est fortement recommandé d'utiliser DLmmx pour tous vos objets issus de la combinaison des primitives du module GD et/ou de vos propres primitives. Mais attention, le résultat peut être altéré si la quadrique englobante est mal positionnée par rapport au volume <*volume*>.
(DLuni <*volume*><*volume*>...)	Cette fonction retourne le volume <*volume*> issu de l'union de Kleene de n volumes <*volume*><*volume*>....

(GDparseg <V4><V4>) Cette fonction retourne le volume <*volume*> formant 2 plans parallèles volumiques passant par 2 points <V4>. Les 2 plans sont perpendiculaires à la droite passant par les 2 points.

(SDcolRGB <V4>) Cette fonction retourne les paramètres de couleur utilisés par la fonction (DLatt) pour associer une couleur à un volume. Les composantes de la couleur sont spécifiés par l'argument <V4>. Ce vecteur est composé des valeurs du rouge, du vert et du bleu dont chacune doit être comprise entre 0 et 1, la 4^{e} coordonnée permet de ramener ces valeurs entre 0 et 1 lorsque l'on travaille avec des tables dont les valeurs dépassent ces bornes (256, 65536...).

(SDmatrep <V4V4>) Cette fonction retourne une matrice utilisée par la fonction (DLatt) comme matrice de changement de repère.

(SGcst_pi) Cette fonction retourne la constante π.

(SGmatmul <VnV4><V4V4>) Cette fonction retourne la matrice <VnV4> issue du produit matriciel entre une matrice <VnV4> et une matrice carrée 4x4 <V4V4>.

(SGmatort <V4>) Cette fonction retourne la matrice carré 4x4 <V4V4> correspondant à la projection orthogonale sur un plan <V4>.

(SGmatrot <V2><V4><V4>) Cette fonction retourne la matrice carré 4x4 <V4V4> correspondant à la rotation d'un angle rationnel en radians <V2> autour d'un axe déterminé par 2 points <V4> et <V4>.

(SGmattrl <V4>) Cette fonction retourne la matrice carré 4x4 <V4V4> correspondant à une translation par rapport à une direction spécifiée par un vecteur <V4>.

(SGqdrhxa <V4><V4V4>) Cette fonction retourne les 6 points de contrôle <V6V4> de la quadrique circonscrite à un hexaèdre projectif défini par le sommet <V4> et le tétraèdre de construction <V4V4>.

(SGvecpro <V4><V4V4>) Cette fonction retourne le vecteur <V4> issu du produit matriciel d'un vecteur <V4> et d'une matrice carrée 4x4 <V4V4>.

(SGx_zx3z <V4><V4><V4>) Cette fonction retourne le point / plan <V4> déterminé par 3 plans / points <V4><V4><V4>.

(SGx__euc <V4>) Cette fonction retourne le point euclidien <V4> [x0/x3 : x1/x3 : x2/x3 : 1] correspondant au point <V4> de coordonnées homogènes [x0 : x1 : x2 : x3]. Les valeurs des coordonnées calculées résultantes peuvent être exprimées sous forme rationnelles.

(SGx___4x <V4><V4><V4><V4>) Cette fonction retourne le point <V4> d'intersection de 2 droites coplanaires définies par 4 points <V4><V4><V4><V4>. Lorsque les droites ne sont pas coplanaires, un point aléatoire est retourné qui peut être le point indéterminé [0 : 0 : 0 : 0].

(SGy2_cir <V3V4>) Cette fonction retourne dans un <V4V4> les 3 sommets du triangle de construction et le point de passage d'une ellipse à partir des 3 sommets <V3V4> du triangle de construction.

	L'ellipse est précontrainte afin d'être circulaire lorsque le triangle est isocèle.
(SGy__coo <V4><V4>)	Cette fonction retourne les coordonnées d'une droite <V6> définie par 2 points <V4> et <V4>.
(SGz__dis <V2><V4>)	Cette fonction retourne le plan <V4> parallèle à une distance donnée <V2> d'un plan <V4>.
(SGz__x_y <V4><V6>)	Cette fonction retourne le plan <V4> défini par un point <V4> et une droite <V6>.

Annexe D

Les fonctions Scheme les plus utilisées

$+$	Addition.
$-$	Soustraction.
$*$	Multiplication.
$/$	Division.
$<$	Inférieur.
$<=$	Inférieur ou égal.
$>$	Supérieur.
$>=$	Supérieur ou égal.

and ET.

or OU.

if Forme spéciale conditionnelle : (if *exp-test exp-alors exp-sinon*).

define

- Nommage d'objets : (define *ident exp*) ; par exemple, après évaluation de l'expression :

  ```
  (define pi 3.14)
  ```

  on a :

  ```
  pi
  => 3.14
  ```

- Définition de fonctions : (define (*nom-fonction x1 x2 ... xk*) *corps*) ; on appelle ensuite cette fonction pour l'évaluer par la commande : (*nom-fonction arg1 arg2 ... argk*).

lambda Définition de procédures sans nom : (lambda (*liste-parametres*) *corps*).

null? Liste vide ?

eq? Equivalent ?

not Non.

apply Applique une procédure à k arguments à une liste à k éléments, comme le montre l'exemple suivant :

```
(apply + '(5 8 2))
```

```
=> 15
```

map Applique une procédure sur les éléments d'une liste et renvoie la liste des résultats :

```
(map * '(1 2 3) (10 10 10))
=> (10 20 30)
```

list Renvoie une liste initialisée avec les arguments entrés :

```
(list '(a b) '(c d) '() '((e)))
=> ((a b) (c d) () (e))
```

cons Retourne une nouvelle liste obtenue en insérant la valeur du premier argument en tête de la liste en second argument :

```
(cons (+ 5 8) '(a b)) => (13 a b)
```

list-tail Renvoie la sous-liste sauf les k preliers éléments.

append Concaténation de listes :

```
(append '(a b) '(c d) '() '((e)))
=> (a b c d (e))
```

reverse Inverse l'ordre des éléments d'une liste.

car Renvoie le premier élément d'une liste.

cdr Renvoie la sous-liste d'une liste sauf le premier élément.

c x1 . . . xk r Composition de **car** et **cdr** (les xj sont des `a` ou des `d`) ; ainsi la fonction composée `car o cdr o cdr` s'écrit aussi `caddr` :

```
(caadr '(a (b) c d e))
=> b
```

list-ref Renvoie le n^{e} élément d'une liste (la numérotation commence à 0) :

```
(list-ref 2 '(a b c d))
=> c
```

length Renvoie le nombre d'éléments d'une liste.

list->vector Renvoie un vecteur initialisé avec les éléments d'une liste.

let Assignation locale, comme par exemple :

```
(let (
      (a 20)
      (b (* 4 8))
      (c 10)
      )
    (* c (- a b))
 )
=> -120
```

let* – Assignation locale en séquence, comme par exemple :

```
(let* (
       (a 4)
       (b (* 3 a))
       (c (* a b))
       )
      (+ a b c)
 )
=> 40
```

– “Named let” : Variante utilisée pour les boucles récursives, dans laquelle la zone de définition des variables correspond à la zone d’intialisation des paramètres de la boucle récursive (incrément, . . .) et celle du corps au corps de la boucle ; **très utilisée dans notre code. . .**

vector	Renvoie un vecteur initialisé avec les arguments entrés.
vector->list	Renvoie une liste initialisée avec les éléments d’un vecteur.
vector-length	Renvoie le nombre d’éléments d’un vecteur.
vector-ref	Renvoie le n^{e} élément d’un vecteur.

load	Chargement d’un programme Scheme.
begin	Evaluation séquentielle d’expressions.
exit	Sortie de l’interpréteur Scheme.

Annexe E

Détail de l'étude analytique de la dent de la crémaillère

Pour l'approche analytique du problème de la génération de la dent de la crémaillère, le schéma de construction est quasiment le même, si ce n'est qu'au lieu de rajouter les arrondis, on les soustrait pour créer la tête de dent.

De plus, comme nous n'en étions qu'à nos débuts sous SGDLsoft, nous n'avons pas utilisé pour définir nos différents composants directement la quadrique et ses 6 points de contrôle, mais des primitives précontraintes (GDprihxa...). Une dernière chose en ce qui concerne l'origine du repère : elle est sur la face avant de la crémaillère, et non sur un plan médian, beaucoup plus intéressant pour étudier le contact, comme c'est le cas sur la version définitive et optimisée du code (présentée en annexe F).

La définition du prisme tronqué (par 2 plans infinis, grâce à la fonction GDparseg) qui va générer notre bloc trapézoïdal ne pose guère de problèmes, il suffit de définir le centre et un sommet de son hexaèdre de construction, l'ordre de ses points de fuite définit son orientation. On n'oubliera pas de définir une direction selon β_0 pour tenir compte du paramètre d'hélice. Pour le détail du code, se reporter à la fin de cet annexe.

Les problèmes commencent à se poser pour définir correctement le cylindre incliné qui va servir à définir l'arrondi en tête de dent : en effet, l'arrondi se situant en réalité dans le plan normal et non dans le plan apparent, les formules de définition des différents points de construction illustrés sur les figures E.1 et E.2 deviennent rapidement très lourdes au niveau trigonométrique, comme on peut le constater dans les formules qui suivent.

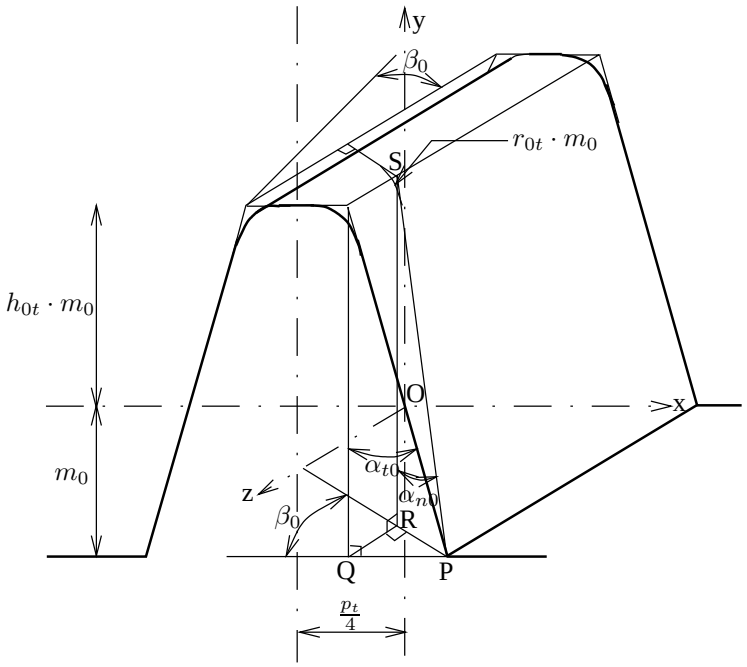


FIG. E.1 – *Géométrie de la dent*

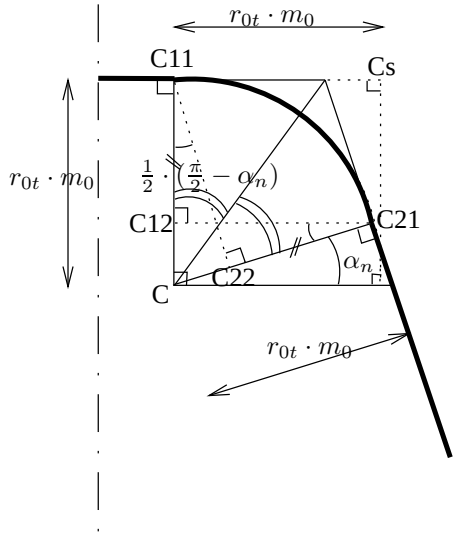


FIG. E.2 – *Vue dans le plan normal de l'arrondi en tête de dent*

Le lecteur pourra constater à quel point ce type de formules est lourd à manipuler et combien la richesse de la géométrie projective peut simplifier le code qui suit en le comparant à celui de l'annexe F.

$$
\begin{aligned}
\vec{OC} &= \vec{OP} + \vec{PQ} + \vec{QR} + \vec{RS} + \vec{SC} \\
&= \begin{pmatrix} m_0 \cdot \tan\alpha_{t0} \\ -m_0 \\ 0 \end{pmatrix} + \begin{pmatrix} -(1+h_{0t}) \cdot m_0 \cdot \tan\alpha_{t0} \\ 0 \\ 0 \end{pmatrix} + \\
&\quad \begin{pmatrix} (1+h_{0t}) \cdot m_0 \cdot \tan\alpha_{t0} \cdot \sin^2\beta_0 \\ 0 \\ -(1+h_{0t}) \cdot m_0 \cdot \tan\alpha_{t0} \cdot \sin\beta_0 \cdot \cos\beta_0 \end{pmatrix} + \begin{pmatrix} 0 \\ (1+h_{0t}) \cdot m_0 \\ 0 \end{pmatrix} + \\
&\quad \begin{pmatrix} -r_{0t} \cdot m_0 \cdot \tan(\frac{1}{2} \cdot (\frac{\pi}{2} - \alpha_{n0})) \cdot \cos\beta_0 \\ -r_{0t} \cdot m_0 \\ -r_{0t} \cdot m_0 \cdot \tan(\frac{1}{2} \cdot (\frac{\pi}{2} - \alpha_{n0})) \cdot \sin\beta_0 \end{pmatrix} \\
&= m_0 \cdot \begin{pmatrix} h_{0t} \cdot \overbrace{\tan\alpha_{t0}}^{\frac{\tan\alpha_{n0}}{\cos\beta_0}} \cdot \overbrace{(\sin^2\beta_0 - 1)}^{-\cos^2\beta_0} + \overbrace{\tan\alpha_{t0}}^{\frac{\tan\alpha_{n0}}{\cos\beta_0}} \cdot \sin^2\beta_0 \overbrace{-}^{-\tan\alpha_{n0}\cdot\cos\beta_0} r_{0t} \cdot \tan(\frac{1}{2} \cdot (\frac{\pi}{2} - \alpha_{n0})) \cdot \cos\beta_0 \overbrace{+}^{\tan\alpha_{n0}\cdot\tan\beta_0\cdot\sin\beta_0} \\ h_{0t} - r_{0t} \\ -((1+h_{0t}) \cdot \tan\alpha_{n0} - r_{0t} \cdot \tan(\frac{1}{2} \cdot (\frac{\pi}{2} - \alpha_{n0}))) \cdot \sin\beta_0 \end{pmatrix} \\
&= m_0 \cdot \begin{pmatrix} \tan\alpha_{n0} \cdot \tan\beta_0 \cdot \sin\beta_0 - (h_{0t} \cdot \tan\alpha_{n0} + r_{0t} \cdot \tan(\frac{1}{2} \cdot (\frac{\pi}{2} - \alpha_{n0}))) \cdot \cos\beta_0 \\ h_{0t} - r_{0t} \\ -((1+h_{0t}) \cdot \tan\alpha_{n0} - r_{0t} \cdot \tan(\frac{1}{2} \cdot (\frac{\pi}{2} - \alpha_{n0}))) \cdot \sin\beta_0 \end{pmatrix} \\
&= \begin{pmatrix} X_C \\ Y_C \\ Z_C \end{pmatrix}
\end{aligned}
$$

De plus, aux vues de ce même code, le lecteur comprendra l'importance du point C : c'est lui qui détermine entièrement la construction de la tête de dent de la crémaillère, tous les autres points ne sont que des projections particulières de ce point : connaissant les coordonnées de C, la détermination de leurs propres coordonnées est triviale.

Voici le code initial de définition d'un pas de la crémaillère, le nom du fichier correspondant est "dent1.scm" :

```
(define dent
  (lambda ()
    (let*
        (
;
;    ---- DEFINITION DES CONSTANTES ----------------------------
;

; Moitie du complementaire de alphan
        (alphac (/ (- (/ pi 2) alphan) 2))
; Tan (alphat)
        (tanat (/ (tan alphan) (cos beta)))

; Sommet du socle
        (S (vector
            (- (/ pt -2) (* mn tanat))
            (- mn) (/ b 2) 1))

; Sommet du bloc prismatique pour generer la dent
        (T (vector
            (- (/ pt -2) (* mn tanat))
            (/  pt (* 4 tanat)) (/ b 2) 1))

; Centre du socle
        (S1 (vector
             (- (* (/ b 2) (tan beta)) (* mn tanat))
             (- (- mn) (/ e 2))
             0 1))

; Centre de l'hexaedre du prisme
        (T1 (vector
             (- (* (/ b 2) (tan beta)) (/ pt 4))
             (- (/ (* pt (cos beta))
                   (* 8 (tan alphan))) (/ mn 2))
             0 1))

; Hexaedre du socle
        (hexas (vector S1
                       (vector (tan beta) 0 -1 0)
                       (vector 1 0 0 0)
                       (vector 0 1 0 0)))

; Hexaedre du prisme
        (hexat (vector T1
                       (vector (tan beta) 0 -1 0)
```

```
                        (vector 1 0 0 0)
                        (vector 0 1 0 0)))

; Coordonnees du point C
        (XC (* mn (- (* (tan alphan) (tan beta) (sin beta))
                     (* (cos beta)
                        (+ (* hot (tan alphan))
                           (* rot (tan alphac)))))))
        (YC (* mn (- hot rot)))
        (ZC (- (* mn (sin beta)
               (- (* (+ 1 hot) (tan alphan))
                  (* rot  (tan alphac))))))

; Projection du point C sur le plan Oxy selon la direction
; indiquee par beta
        (XCp (+ XC (* ZC (tan beta))))
        (YCp YC)
        (ZCp (/ b 2))

; Point Cs servant a determiner le sommet de l'hexaedre
; de construction du cylindre de l'arrondi
        (XCs (+ XC (* mn rot (cos beta))))
        (YCs (+ YC (* mn rot)))
        (ZCs (+ ZC (* mn rot (sin beta))))

; Sommet de l'hexaedre du cylindre des rayons (projection
; du point Cs sur le plan Oxy selon la direction beta)
        (Csp (vector
              (+ XCs (* ZCs (tan beta)))
              YCs (/ b 2) 1))

; Centre de l'hexaedre du cylindre des rayons
        (Cc (vector
             (+ XCp (* (/ b 2) (tan beta)))
             YCp
             0 1))

; Hexaedre de l'arrondi
        (hexac (vector Cc
                       (vector 1 0 0 0)
                       (vector 0 1 0 0)
                       (vector (tan beta) 0 -1 0)))

        (XC11 XC)
        (YC11 (+ YC (* mn rot)))
        (ZC11 ZC)

        (XC12 XC)
        (YC12 (+ YC (* mn rot (tan alphan))))
```

```
         (ZC12 ZC)

         (XC21 (+ XC (* mn rot (cos alphan)
                       (cos beta))))
         (YC21 (+ YC (* mn rot (sin alphan))))
         (ZC21 (+ ZC (* mn rot (cos alphan)
                     (sin beta))))

         (XC22 (+ XC (* mn rot (sin alphan)
                     (cos alphan) (cos beta))))
         (YC22 (+ YC (* mn rot (sin alphan)
                     (sin alphan))))
         (ZC22 (+ ZC (* mn rot (sin alphan)
                     (cos alphan) (sin beta))))
          )

;
;    ---- CORPS DU PROGRAMME -----------------------------------
;
      (let
          (
; Creation du profil a enlever a droite pour generer l'arrondi
; en faisant la ; difference entre un volume genere par des
; plans paralleles  2 a 2 definissant les points de tangence
; C11 et C21 et le cylindre de centre Csp et d'hexaedre hexac
           (arrondi
            (DLdif
             (DLint
              (GDparseg
               (vector XC11 YC11 ZC11 1)
               (vector XC12 YC12 ZC12 1))
              (GDparseg
               (vector XC21 YC21 ZC21 1)
               (vector XC22 YC22 ZC22 1))
              (GDparseg
               (vector 0 0 (/ b -2) 1)
               (vector 0 0 (/ b 2) 1)))
             (GDcylhxa Csp hexac)))
            )
; Creation de la cremaillere
         (DLdif
          (DLdif
           (DLuni
            (GDhxahxa S hexas)
            (DLint
             (GDprihxa T hexat)
             (GDparseg
              (vector 0 (* mn hot) 0 1)
              (vector 0 (- mn) 0 1)))))
```

```
; Creation du volume total a enlever pour generer les arrondis de
; tete de dent de la cremaillere en faisant une union du profil
; "arrondi" et de son symetrique par rapport a une droite de
; direction Oy et passant par le centre de l'hexaedre de
; construction de la dent
         (DLuni
          arrondi
          (DLatt
           (SDmatrep
            (SGmatsmy
             (SGy__coo
              (vector (- (* (/ b 2) (tan beta)) (/ pt 4)) 0 0 1)
              (vector (- (* (/ b 2) (tan beta)) (/ pt 4)) 1 0 1)
              )))
            arrondi))))
)))
```

Annexe F

Détail du code de l'approche "taillage"

Voici le code de définition d'une dent de la crémaillère de taillage (pour les différents paramètres, le lecteur se reportera aux pages de notations en début de rapport et aux figures 4.1 et 4.2), le nom du fichier correspondant est "dent-crem.scm" :

```
(define dent-crem
  (lambda (m0 alpha0 beta0 b)
    (let*
        (
;
;    ---- DEFINITION DES CONSTANTES ----------------------------
;
         (pi (SGcst_pi))
         (hot 1.25)
         (rot 0.35)

         (pt (/ (* pi m0) (cos beta0)))
         (mt0 (/ m0 (cos beta0)))
         (alphat0 (atan (/ (tan alpha0) (cos beta0))))
;
;    ---- DEFINITION DES SOUS-FONCTIONS ------------------------
;
; Fonction de projection dans le plan ‘‘plan’’ d’un point ‘‘point’’
; selon une direction ‘‘direction’’
         (projx
          (lambda (plan point direction)
            (SGx__y_z plan (SGy__coo point direction))))

; Fonction de translation selon une distance ‘‘distance’’ d’un point
; ‘‘point’’ dans la direction ‘‘direction’’
         (translx
          (lambda (distance point direction)
            (SGx__dis distance point direction)))

; Fonction de generation de la quadrique englobante du volume ‘‘volume’’
; circonscrite a l’hexaedre ‘‘hexaedre’’
```

```
        (min-max
         (lambda (hexaedre volume)
           (DLmmx
            (DLformz2
             (apply SGqdrhxa hexaedre))
            volume)))
;
;    ---- DEFINITION DES POINTS DE REFERENCE DANS P0 -----------
;
        (A0 (vector (+ (/ pt -2) (* -1 m0 (tan alphat0))) (- m0) 0 1))
        (B0 (vector (* m0 (tan alphat0)) (- m0) 0 1))
        (C0 (vector (- (* hot m0 (tan alphat0))) (* hot m0) 0 1))
        (Q0 (vector (/ pt -4) (- m0) 0 1))
        (R0 (vector (/ pt -4) 0 0 1))
        (S0 (vector (/ pt -4) (/ pt 4 (tan alphat0)) 0 1))
        (T0 (vector (/ pt -4) (* hot m0) 0 1))
;
;    ---- DEFINITION DES DIRECTIONS ET PLANS ------------------
;
; On remarquera l'utilisation de l'indice 0 pour les points en z=0, n pour
; les points dans le plan Pn (plan normal de la cremaillere passant par I0),
; 1 pour les points de la face avant de la cremaillere (z=b/2) et 2 pour la
; face arriere (z=-b/2)
        (I0 (vector 0 0 0 1))
        (I1 (vector 0 0 (/ b 2) 1))
        (I2 (vector 0 0 (/ b -2) 1))

        (X0 (vector 1 0 0 0))
        (Y0 (vector 0 1 0 0))
        (Z0 (vector 0 0 1 0))

        (Xn (vector (cos beta0) 0 (sin beta0) 0))
        (Yn Y0)
        (Zn (vector (- (sin beta0)) 0 (cos beta0) 0))

        (P0 Z0)
        (P1 (vector 0 0 1 (/ b -2)))
        (P2 (vector 0 0 1 (/ b 2)))
        (Pn Zn)

        (dir-alpha0 (vector (* (- (sin alpha0)) (cos beta0)) (cos alpha0)
                            (* (- (sin alpha0)) (sin beta0)) 0))
;
;    ---- CALCUL DES POINTS UTILES ----------------------------
;
; Projection dans le plan P1 du point S0 selon la direction Zn...
        (S1 (projx P1 S0 Zn))
        (C1 (projx P1 C0 Zn))
        (T1 (projx P1 T0 Zn))
```

```
        (Q1 (projx P1 Q0 Zn))
        (B1 (projx P1 B0 Zn))

; Definition du plan median de la dent (plan passant par R0, T0 et T1)
        (plan_sym (SGx_zx3z R0 T0 T1))

; Distance entre Vn et Cn (dans le plan Pn)
        (dist (* m0 rot (tan (/ (- (/ pi 2) alpha0) 2))))

        (Cn (projx Pn C0 Zn))
; Definition du point Un par translation du point Cn d'une distance
; dist dans la direction - dir-alpha0
        (Un (translx (vector dist -1) Cn dir-alpha0))
        (Vn (translx (vector dist -1) Cn Xn))
        (Wn (vector-ref (SGy2_cir (vector Un Vn Cn)) 3))
; Definition du point Usymn comme le symetrique de Un par rapport au
; plan median plan_sym (donc dans le plan Pn)
        (Usymn (SGvecpro Un (SGmatsmz plan_sym)))
        (Vsymn (SGvecpro Vn (SGmatsmz plan_sym)))

        (U0 (projx P0 Un Zn))
        (U1 (projx P1 Un Zn))
        (V1 (projx P1 Vn Zn))

        (KUVn (SGx_zx3z Pn (SGx_zx3z Un U1 Xn)  (SGx_zx3z Vn V1 Yn)))
        (KUVsymn (SGvecpro KUVn (SGmatsmz plan_sym)))

        (Ux0 (projx plan_sym U0 X0))

        (quadric-prisme (vector A0 B0 S0 S1 S0 S1))
        (quadric-raccord1 (vector Un Vn Cn Zn Wn V1))
        (quadric-raccord2 (SGvcxsmz quadric-raccord1 plan_sym))
;
;    ---- DEFINITION DES VOLUMES ---------------------------
;
        (prisme
; Intersection de la quadrique prisme avec les volumes compris entre
; 2 plans paralleles definis comme ci-dessous pour obtenir un bloc
; parallelepipedique fini et a base trapezoidal
         (DLint
          (DLformz2 quadric-prisme)
          (GDparseg Q0 Ux0)
          (GDparseg I1 I2)
          ))

        (boite
         (GDparseg Vn Vsymn))

        (raccord1
```

```
          (DLint
           (DLformz2 quadric-raccord1)
           (GDparseg Un KUVn)
           ))

         (raccord2
          (DLint
           (DLformz2 quadric-raccord2)
           (GDparseg Usymn KUVsymn)
           ))
;
;    ---- DEFINITION DES QUADRIQUES ENGLOBANTES ----------------
;
; Definition du sommet (B1), du centre (milieu du segment Q0T0) et des
; points de fuite (X0, Y0 et Zn, tous a l'infini, donc ce sont aussi des
; directions...) de l'hexaedre de construction de la quadrique englobante
; de la dent
         (mmx-dent
          (list
           B1
           (vector
            (SGx__med Q0 T0) X0 Y0 Zn)))

         (mmx-prisme
          (list
           B1
           (vector
            (SGx__med Q0 Ux0) X0 Y0 Zn)))

         (mmx-raccords
          (list
           U1
           (vector
            (SGx__med T0 Ux0) X0 Y0 Zn)))

         )
;
;    ---- CORPS DE LA FONCTION DENT-CREM -----------------------
;
       (min-max
        mmx-dent
        (DLadd
         (min-max mmx-prisme prisme)
         (min-max
          mmx-raccords
          (DLint
           (DLadd
            raccord1
            raccord2
```

```
           boite
           )
          (GDparseg I1 I2)
          (GDparseg T0 Ux0))
          )
        ))
      )))
```

A présent, à partir de cette fonction `dent-crem`, nous allons passer au code de génération d'un engrenage et de la crémaillère de taillage (on a adopté l'algorithme de translation de la crémaillère et de rotation de la roue), le nom du fichier correspondant est "engren-crem.scm". Il est bon de noter l'optimisation de la quadrique englobante de la crémaillère (on ne considère que la partie utile ; pour le point P dans le code, se reporter à la figure 6.2).

```
;-------------------------------------------------------------
;   GENERATION DE L'ENGRENAGE
;   Les parametres sont:
;     - m0 => module reel
;     - alpha0 => angle de pression reel (en degres)
;     - beta0 => angle d'inclinaison d'helice (en degres)
;     - b => largeur de la cremaillere
;     - Z => nombre de dents souhaite
;     - tetamax => portion angulaire sur laquelle on veut
;                  representer les dents
;     - k => increment decoupant tetamax en k intervalles
;                  (precision du trace)
;     - rotatZ => Angle de rotation de l'engrenage autour de
;                  l'axe Z
;     - rotatY => Angle de rotation de l'engrenage autour de
;                  l'axe Y
;-------------------------------------------------------------

(define engrenage
  (lambda (m0 alpha0 beta0 b Z tetamax k rotatZ rotatY)
    (let*
        (
;
;    ---- DEFINITION DES CONSTANTES ----------------------------
;
         (pi (SGcst_pi))
         (hot 1.25)
         (rot 0.35)
; Epaisseur du socle de la cremaillere
         (e 10)

         (d (/ (* m0 Z) (cos beta0)))
```

```
        (r (/ (* m0 Z) (cos beta0) 2))
        (pt (/ (* pi m0) (cos beta0)))
        (mt0 (/ m0 (cos beta0)))
        (alphat0 (atan (/ (tan alpha0) (cos beta0))))
; Coordonnees polaires du point P
        (x-intersec  (sqrt (- (* 4.5 m0 r) (/ (sqr m0) 16))))
        (teta-intersec
         (asin (/ (sqrt (- (* 4.5 m0 r) (/ (sqr m0) 16))) (+ r m0) -1)))

; Definition du pas angulaire a partir de k
        (pas_ang (/ tetamax k))
; Nombre de dents maximum de taillage (en fonction de l'angle sur
; lequel on taille (tetamax) (ceiling est une fonction qui prend
; la partie entiere par valeur superieure)
        (Zmax (ceiling (/ (* Z tetamax) 2 pi)))
;
;    ---- DEFINITION DES DIRECTIONS ----------------------------
;
        (I0 (vector 0 0 0 1))
        (X0 (vector 1 0 0 0))
        (Y0 (vector 0 1 0 0))
        (Z0 (vector 0 0 1 0))

        (Zn (vector (- (sin beta0)) 0 (cos beta0) 0))
;
;    ---- DEFINITION DES SOUS-FONCTIONS ------------------------
;
        (min-max
         (lambda (hexaedre volume)
           (DLmmx
            (DLformz2
             (apply SGqdrhxa hexaedre))
            volume)))

; Fonction d'attribution d'une couleur a un volume
        (color
         (lambda (color volume)
           (DLatt
            (SDcolRGB color)
            volume)))
;
;    ---- DEFINITION DES COULEURS ------------------------------
;
        (gris-jaune (vector 1 1 .6 1))
        (jaune (vector 1 1 0 1))
        (vert (vector 0 1 1 1))
        (bleu-metal (vector 0 .5 .7 1))
        (rouge (vector 1 0 0 1))
```

```
;-------------------------------------------------------------
;   FONCTION DE TAILLAGE
;   Les parametres sont:
;     - roue => roue a tailler
;     - cremaillere => cremaillere
;     - i => increment
;     - mmx => min-max de la cremaillere
;
; On fait tourner la roue de l'increment et on lui enleve la
; cremaillere
;-------------------------------------------------------------

         (taille
          (lambda  (roue cremaillere i mmx)
            (DLatt
             (SDmatrep
              (SGmatrot
               (vector pas_ang 1)
               (vector 0 r 0 1)
               (vector 0 r 1 1)))
             (DLdif
              roue
              (min-max
               mmx
               (DLatt
                (SDmatrep
                 (SGmattrl
                  (vector  (* r i pas_ang) 0 0 1)))
                cremaillere))
              ))
          ))

;-------------------------------------------------------------
;   FONCTION TRACER
;   Les parametres sont:
;     - roue => roue taillée
;     - cremaillere => cremaillere de Z dents
;     - mmx1 => min-max de la roue taillée
;     - mmx2 => min-max de la crémaillère
; On fait tourner la roue de 90 degres autour de Y ou de 180
; autour de Z
;-------------------------------------------------------------

         (tracer
          (lambda (roue cremaillere mmx1 mmx2)
            (DLatt
             (SDmatrep
              (SGmatmul
               (SGmatrot
```

```
                (vector rotatY 1)
                I0 Y0)
               (SGmatrot
                (vector rotatZ 1)
                I0 Z0)))
             (DLuni
              (color vert (min-max mmx1 roue))
              (color
               bleu-metal
               (DLatt
                (SDmatrep
                 (SGmattrl
                  (vector (* r tetamax) 0 0 1)))
                 (min-max  mmx2 cremaillere))
                ))
             )))

;
;    ---- DEFINITION DES MIN-MAX -------------------------------
;
; Quadrique englobante pour la roue
        (mmx-roue
         (list
          (vector (+ r m0) (- m0) (/ b 2) 1)
          (vector
           (vector 0 r 0 1)
                   X0 Y0 Z0)))

; Quadrique englobante pour la cremaillere de Zmax dents
        (mmx-cremaillere
         (list
          (vector (- (* m0 (tan alphat0)) (/ (* b (tan beta0)) 2))
                  (- 0 e m0) (/ b 2) 1)
          (vector
           (vector (/ (* (+ 0.5 Zmax) pt) -2)
                   (/ (- (* hot m0) m0 e) 2) 0 1)
                   X0 Y0 Zn)))

; Quadrique englobante pour la cremaillere de taillage
        (mmx-cremaillere-taillage
         (list
          (vector x-intersec (- m0) (/ b 2) 1)
          (vector
           (vector 0 (/ (* (- hot 1) m0) 2) 0 1)
                   X0 Y0 Zn)))
;
;    ---- DEFINITION DES VOLUMES -------------------------------
;
; Creation d'une dent de la cremaillere par appel de la fonction
```

```
; ''dent-crem''
        (modele (dent-crem m0 alpha0 beta0 b))

; Cylindre avec une rainure de clavette
        (cylin
         (min-max
          mmx-roue
          (DLdif
           (GDcylseg
            (vector (+ r m0) 1)
            (vector 0 r (/ b -2) 1)
            (vector 0 r (/ b 2) 1))
           (GDcylseg
            (vector r 5)
            (vector 0 r (/ b -2) 1)
            (vector 0 r (/ b 2) 1))
           (GDhxahxa
            (vector (/ r 12) (/ (* 15 r) 12) (/ b 2) 1)
            (vector
             (vector 0 (/ (* 6 r) 5) 0 1)
                    X0 Y0 Z0)))))

; Generation de la cremaillere de Zmax dents par une fonction recursive
        (cremaillere
         (let repet ((list-dent (list))
                     (Zi 0))
           (if (<= Zi Zmax)
  ; Generation du nombre adequat de dents et placement de celles-ci (on
  ; stocke le tout dans une liste ''list-dent'')
               (repet (cons
                       (DLatt
                        (SDmatrep
                         (SGmattrl
                          (vector  (* -1 Zi pt) 0  0 1)))
                         modele) list-dent)
                      (+ Zi 1))
  ; Une fois le bon nombre de dents obtenu, on ajoute le socle de la
  ; cremaillere (qu'on stocke dans la meme liste)
               (DLadd
                (apply DLuni list-dent)
                (GDhxahxa
                 (vector (- (* m0 (tan alphat0)) (/ (* b (tan beta0)) 2))
                         (- m0) (/ b 2) 1)
                 (vector
                  (vector (/ (* (+ .5 Zmax) pt) -2) (- 0 m0 (/ e 2)) 0 1)
                   X0 Y0 Zn)))
             )))

        )
```

```
;
;    ---- CORPS DU PROGRAMME ----------------------------------
;
       (let boucle ((roue cylin)
                    (i 0))
         (if (<= i (- k 1))
             (boucle
              (taille roue cremaillere i mmx-cremaillere-taillage)
              (+ i 1))
             (tracer roue cremaillere mmx-roue mmx-cremaillere)
           ))
;
;    ---- FIN DU CORPS DU PROGRAMME ---------------------------
;
 )))
```

Ce code est intéressant, mais comme on trace la crémaillère en même temps que l'engrenage, le calcul de la partie réellement utile de taillage à chaque incrément n'est pas fait puisqu'il nous faut de toute façon afficher toute la crémaillère.

Dans le code qui suit (le nom du fichier est "engrenage.scm" sont présentées les modifications à faire pour optimiser au mieux le code au niveau de la crémaillère, afin de ne calculer que la partie utile à chaque itération (se reporter au paragraphe 6 pour voir la détermination de la zone utile sur la crémaillère). On notera aussi que comme on n'affiche pas la crémaillère, il est inutile de tenir compte du socle (puisque notre roue a un diamètre égal au diamètre de tête de l'engrenage).

```
; Generation de la cremaillere de taillage par une fonction recursive
; Les parametres sont:
;   - i => increment definissant la position de taillage
; (floor fait l'inverse de ceiling, cette fonction prend la partie
;  entiere par valeur inferieure)
        (cremaillere-taillage
         (lambda (i)
           (let repet ((list-dent (list))
  ; Pour aleger l'arbre volumique, on calcule a chaque taillage la
  ; portion utile de cremaillere. Pour cela on calcule la dent
  ; correspondant a l'iteration et on fabrique une petite cremaillere
  ; de 5 dents (-2 a gauche et + 2 a droite)
                       (Zi (- (floor (/ (* Zmax i) k)) 2)))
             (if (<= Zi (+ (floor (/ (* Zmax i) k)) 2))
                 (repet (cons
                         (DLatt
                          (SDmatrep
                           (SGmattrl
                            (vector  (* -1 Zi pt) 0  0 1)))
   modele) list-dent)
```

```
                        (+ Zi 1))
                    (apply DLuni list-dent))
                )))
;
;    ---- CORPS DU PROGRAMME ----------------------------------
;
      (let boucle ((roue cylin)
                   (i 0))
        (if (<= i (- k 1))
            (boucle (taille
                      roue
                      (cremaillere-taillage i)
                      i
                      mmx-cremaillere-taillage)
                    (+ i 1))
            (tracer roue mmx-roue)
         ))
;
;    ---- FIN DU CORPS DU PROGRAMME ---------------------------
;
```

Pour conclure sur cette optimisation du code, on peut dire que, schématiquement, dans le premier cas la crémaillère calculée a pour longueur celle de l'arc sur lequel on veut tailler, alors que dans le dernier cas, la crémaillère a pour longueur celle de l'hexaèdre de sa quadrique englobante. D'ailleurs, il est bon de noter que dans les deux cas cette quadrique est la même : en effet, avoir une quadrique plus petite permettra tout de même d'optimiser le calcul au niveau des intersections de la crémaillère avec la roue, mais au tracé il faut absolument prendre la crémaillère entière.

Un dernier changement au niveau de ce code : comme on ne trace qu'un seul volume (l'engrenage), on peut se passer de l'affectation des couleurs à ce niveau du code, on les retrouvera plutôt dans le fichier de visualisation ("visu-engrenage.scm") qui suit.

```
;
;    ---- PARAMETRES DE VISUALISATION -------------------------
;
(define setproj
  (lambda (OX OY OZ resolution zoom b r1)
    (begin
; Initialisation des parametres d'affichage
      (PIreset)
; Definition de la position de l'observateur
      (PIsobs (vector OX OY OZ 0))
; Definition de la position du point vise
      (PIstarget (vector 0 20 0 1))
; Definition du zoom (inutile si en perspective)
      (PIsbox zoom)
```

```
; Definition de la resolution (800x600...)
      (PIswin resolution)
; Definition de la couleur du fond de l'ecran
      (PIswincol (vector 1 1 0.9))

; Ajout de lumieres annexes et leurs differents parametres
      (PIalight (vector (vector r1 (* 2.5 r1) 0 1) (vector 7)
                        (vector 0.1 0.1 0.2 1 1 1 3 3 3)
                        (vector 1 1 1 1 1 1 1 1 1 )))
      (PIalight (vector (vector 0 (/ r1 2) b .5) (vector 7)
                        (vector 0.1 0.1 0.2 1 1 1 3 3 3)
                        (vector 1 1 1 1 1 1 1 1 1 )))
     )))

;
;    ---- LANCEMENT DU TRACE ----------------------------------
;
(define tracer
  (lambda ()
    (let*
        (
         (m0 6)
         (alpha0 20)
         (beta1 30)
         (b 100)
         (Z1 20)
         (tetamax 90)
         (k 10)
         (rotatZ1 0)
         (rotatY1 0)

         (r1 (/ (* m0 Z1) (cos (rad beta1)) 2))
         (I (vector 0 0 0 1))
         (O (lambda (r) (vector  0 r 0 1)))
         (Vx (vector 1 0 0 0))
         (Vy (vector 0 1 0 0))
         (Vz (vector 0 0 1 0))

         (resolution (vector 640 480))
         (zoom (vector 300 300))
         (OX .3)
         (OY .4)
         (OZ 1)

 (color
  (lambda (color volume)
    (DLatt
     (SDcolRGB color)
     volume)))
```

```
        (symetrie
         (lambda (volume)
           (DLatt
            (SDmatrep
             (SGmatsmx I))
             volume)))

        (rotation
         (lambda (r rotatY rotatZ volume)
           (DLatt
            (SDmatrep
             (SGmatmul
              (SGmatrot (vector (rad rotatY) 1) I Vy)
              (SGmatrot (vector (rad rotatZ) 1) (O r)  Vz)))
            volume)))

; Definition des couleurs
        (gris-jaune (vector 1 1 .6 1))
        (jaune (vector 1 1 0 1))
        (vert (vector 0 1 1 1))
        (bleu-metal (vector 0 .5 .7 1))
        (rouge (vector 1 0 0 1))
        )

; Chargement des fichiers de generation de la dent de la cremaillere
; et de generation de l'engrenage
      (load "dent-crem.scm")
      (load "engrenage.scm")
; Appel des parametres d'affichage
      (setproj OX OY OZ resolution zoom b r1)
; Calcul de la scene volumique a afficher et stockage sous le nom
; ''engrenage''
      (PIgen
       (DLuni
        (color vert
               (engrenage m0 (rad alpha0) (rad beta1) b Z1 (rad tetamax)
                          k (rad rotatZ1) (rad rotatY1)))
        )
       "engrenage")
; Affichage de la scene volumique et stockage de la vue sous le nom
; ''engrenage''
      (PIview "engrenage" "engrenage")
; Exportation de la vue affichee en format .tga sous le nom
; ''engrenage_{valeur du module}-{valeur de beta1}-{valeur de Z1}
; -{valeur de b}-{valeur de k}.tga''
      (PIshow->targa "engrenage"
                     (string-append "engrenage_"
                                    (number->string m0)
```

```
                                        "-"
                                        (number->string beta1)
                                        "-"
                                        (number->string Z1)
                                        "-"
                                        (number->string b)
                                        "-"
                                        (number->string k)
                              ))
      )
    ))

(tracer)
```

Annexe G

Détail du code de l'approche mathématique

Dans un premier temps, nous allons présenter le code traitant de l'approximation linéaire de la denture (par des trapèzes), le fichier source est nommé "dent-trapeze.scm". Pour le détail des notations, elles suivent les notations ISO déjà présentées au début de ce rapport, sinon pour plus de renseignements se reporter aux travaux de M. LAURIA [10].

```
;-------------------------------------------------------------
;   GENERATION D'UN BLOC TRAPEZOIDAL
;   Les parametres sont:
;     - A1       => point extremite de la grande base du trapeze
;     - A2       => point extremite de la petite base du trapeze
;     - A3       => introduisant la 3eme dimension (pour
;                   l'extrusion selon une direction donnee du
;                   trapeze)
;     - plan_sym => plan passant par l'axe de symetrie
;                   perpendiculaire aux bases du trapeze
;-------------------------------------------------------------

(define trapeze
  (lambda (A1 A2 A3 plan_sym)
    (let*
        (
;
;    ---- DEFINITION DES CONSTANTES ----------------------------
;
; Definition des points de controle de la quadrique prisme qui va
; servir a generer une boite a base trapezoidale
         (P0 A1)
         (P1 (SGvecpro A1 (SGmatsmz plan_sym)))
         (P1bis (SGvecpro A2 (SGmatsmz plan_sym)))
         (P2 (vector 0 0 1 0))
         (P3 (SGx___4x A1 A2 P1 P1bis))
         (P4 P2)
         (P5 P3)
 )
;
```

```
;    ---- CORPS DU PROGRAMME ----------------------------------
;
; Generation d'un bloc trapezoidal fini
       (DLint
        (GDparseg
         (vector 0 0 (vector-ref A1 2) 1)
         (vector 0 0 (vector-ref A3 2) 1))
        (GDparseg
         (SGvecpro A1 (SGmatort plan_sym))
         (SGvecpro A2 (SGmatort plan_sym)))
        (DLformz2 (vector P0 P1 P2 P3 P4 P5))
        ))
   ))
;
;    ---- FIN DU CORPS DU PROGRAMME ---------------------------
;

;-------------------------------------------------------------
;   GENERATION DE LA DENTURE EN HELICE
;   Les parametres sont:
;     - m0 => module reel
;     - alpha0 => angle de pression reel (en degres)
;     - beta0 => angle d'inclinaison d'helice (en degres)
;     - bh => facteur multiplicatif pour la largeur de la
;                 denture (b = bh . m0)
;     - Z1 => nombre de dents souhaite sur l'engrenage 1
;     - Z1 => nombre de dents souhaite sur l'engrenage 2
;     - x1 => deport sur la denture 1
;     - x2 => deport sur la denture 2
;     - inceps => increment sur le parametre epsilon (eps)
;     - incZ => increment sur le parametre Z (extrusion)
;-------------------------------------------------------------

(define dent
  (lambda (m0 alpha0 beta0 bh Z1 Z2 x1 x2 inceps incZ)
    (let*
        (
;
;    ---- FONCTION DIVERSES -----------------------------------
;
; Applique une couleur sur un volume
         (color
          (lambda (color volume)
            (DLatt
             (SDcolRGB color)
             volume)))

; Applique un min-max sur un volume
         (min-max
```

```
          (lambda (hexaedre volume)
            (DLmmx
             (DLformz2
              (apply SGqdrhxa hexaedre))
             volume)))

; Ces fonctions renvoient la 1ere et la 2e coordonnee d'un vecteur
        (Vx  (lambda (ls) (map (lambda (V) (vector-ref V 0)) ls)))
        (Vy  (lambda (ls) (map (lambda (V) (vector-ref V 1)) ls)))
;
;    ---- DEFINITION DES COULEURS -----------------------------
;
        (gris-jaune (vector 1 1 .6 1))
        (jaune (vector 1 1 0 1))
        (vert (vector 0 1 1 1))
        (bleu-metal (vector 0 .5 .7 1))
        (rouge (vector 1 0 0 1))
;
;    ---- DEFINITION DE CONSTANTES ----------------------------
;
        (pi (SGcst_pi))
        (O (vector 0 0 0 1))
        (OX (vector 1 0 0 0))
        (OY (vector 0 1 0 0))
        (OZ (vector 0 0 1 0))
;
;    ---- DEFINITION DES FONCTIONS MATHEMATIQUES --------------
;
; Fonction involute
        (inv (lambda (alpha) (- (tan alpha) alpha)))

; Fonction derivée de l'involute (dinv/dalpha)
        (invp (lambda (alpha) (sqr (tan alpha))))

; Fonction réciproque de l'involute (methode de Newton-Raphson)
        (arcinv
         (lambda (teta)
            (let repet (t (min 1.57 (expt (* 3 teta) (/ 3))))
             (if
              (or
               (> t 1.57) (< (abs (- (inv t) teta)) (/ (^ 10 10))))
              t
              (repet
               (- t (/ (- (inv t) teta) (invp t))))
              ))))
;
;    ---- DEFINITION DES PARAMETRES MECANIQUES (LAURIA) --------
;
        (mt0 (/ m0 (cos beta0)))
```

```
(alphat0 (atan (/ (tan alpha0) (cos beta0))))
(r1 (/ (* mt0 Z1) 2))
(r2 (/ (* mt0 Z2) 2))

(alphapt (arcinv (+ (inv alphat0)
                    (/ (* 2 (+ x1 x2) (tan alpha0))
                       (+ Z1 Z2)))))

(rp1 (/ (* (cos alphat0) r1) (cos alphapt)))
(rp2 (/ (* (cos alphat0) r2) (cos alphapt)))
(mpt (/ (* (cos alphat0) mt0) (cos alphapt)))
(ap (+ rp1 rp2))
(betap (atan (* (/ rp1 r1) (tan beta0))))
(mpn (* mpt (cos betap)))
(b (* bh mpn))
(Ppn (* pi mpn))
(betab (atan (* (tan betap) (cos alphapt))))
(rb1 (* r1 (cos alphat0)))
(rb2 (* r2 (cos alphat0)))
(S1 (* mt0 (+ (/ pi 2) (* 2 x1 (tan alpha0)))))
(Sb1 (* rb1 (+ (/ S1 r1) (* 2 (inv alphat0)))))
(Sp1 (* rp1 (- (/ Sb1 rb1)
               (* 2 (inv (acos (/ rb1 rp1)))))))

(S2 (* mt0 (+ (/ pi 2) (* 2 x2 (tan alpha0)))))
(Sb2 (* rb2 (+ (/ S2 r2) (* 2 (inv alphat0)))))
(Sp2 (* rp2 (- (/ Sb2 rb2)
               (* 2 (inv (acos (/ rb2 rp2)))))))

(alphapn (atan (* (tan alphapt) (cos betap))))
(Pbn (* Ppn (cos alphapn)))
(Pbt (/ Pbn (cos betab)))
(K  (* (/ (+ Z1 Z2) 2)
       (- (/ (* 2 (+ x1 x2)) (+ Z1 Z2))
          (/ (- (/ (cos alphat0) (cos alphapt)) 1)
             (cos beta0)))))
(ra1 (* m0 (+ (/ Z1 (cos beta0) 2) (+ 1 x1) (- K))))
(ra2 (* m0 (+ (/ Z2 (cos beta0) 2) (+ 1 x2) (- K))))
(ha1 (- ra1 rp1))
(ha2 (- ra2 rp2))
(h (* m0 (- 2.25 K)))
(hf1 (- h ha1))
(hf2 (- h ha2))

(phi1 (sqrt
       (- (sqr (/ (+ Z1 (/ (* 2 ha1 (cos betap)) mpn))
                  Z1 (cos alphapt))) 1)))
(phi2 (sqrt
       (- (sqr (/ (+ Z2 (/ (* 2 ha2 (cos betap)) mpn))
```

```
                    Z2 (cos alphapt))) 1)))
        (phif (/ (* Z2 (- (* phi2 (cos alphapt)) (sin alphapt)))
                 2 pi (cos betap)))
        (phia (/ (* Z1 (- (* phi1 (cos alphapt)) (sin alphapt)))
                 2 pi (cos betap)))
        (phibeta (/ (sin betab) pi (cos alphapn)))

        (gf (* phif Ppn))
        (ga (* phia Ppn))
        (gbeta (* b (tan betab)))

        (ractif1 (sqrt (+ (* rb1 rb1 (+ 1 (sqr (tan alphapt))))
                          (sqr gf) (* -2 rb1 gf (tan alphapt)))))
        (ractif2 (sqrt (+ (* rb2 rb2 (+ 1 (sqr (tan alphapt))))
                          (sqr ga) (* -2 rb2 ga (tan alphapt)))))

        (Jeps (/ (* pi (cos alphapn) mpn) (cos betab)))
        (Jy (+ (* pi phif mpn) (/ (* b (tan betab)) 2)))

        (epsalpha (/ (+ gf ga) Pbt))
        (epsbeta (/ (* b (tan betab)) Pbt))
        (epsgamma (+ epsalpha epsbeta))

        (epsI (+ (/ (* phif (cos betab)) (cos alphapn))
                 (/ epsbeta 2)))

        (paseps (/ (+ ga gf) inceps Jeps))
        (pasZ (/ b incZ 2))

        (K1 (* (tan betab) (sin alphapt)))
        (K2 (* (tan betab) (cos alphapt)))
        (K3 (* (sin alphapt) Jeps))
        (K4 (* (cos alphapt) Jeps))
        (K5 (- (* (sin alphapt) Jy)))
        (K6 (- (* (cos alphapt) Jy)))

        (epsilonp1 (/ (* rb1 epsgamma) (+ gf ga gbeta)))

        (Zmin (/ b -2))
        (Zmax (/ b 2))
;
;    ---- DEFINITION DES SOUS-FONCTIONS ------------------------
;
        (psi (lambda (Z) (+ (/ Sp1 -2 rp1) (/ (* gbeta Z) b rb1))))

        (epsf (lambda (Z) (/ (* (- (/ b 2) Z) (tan betab)) Jeps)))

        (epsa (lambda (Z) (/ (+ (* 2 Ppn phif)
                                (* (- (/ b 2) Z) (tan betab))) Jeps)))
```

```
          (X_psi
           (lambda (Z)
             (vector (cos (psi Z)) (sin (psi Z)) 0 0)))

          (Y_psi
           (lambda (Z)
             (vector (- (sin (psi Z))) (cos (psi Z)) 0 0)))

          (plan_sym
           (lambda (Z)
             (vector (- (sin (psi Z))) (cos (psi Z)) 0 0)))

          (lamb (lambda (eps) (/ (* (+ gf ga gbeta) (- eps epsI))
                                 rb1 epsgamma)))

          (Qxz
           (lambda (eps)
             (+ (* K1 (cos (lamb eps))) (* K2 (sin (lamb eps))))))
          (Qxeps
           (lambda (eps)
             (+ (* K3 (cos (lamb eps))) (* K4 (sin (lamb eps))))))
          (Qx
           (lambda (eps)
             (+ (* rp1 (cos (lamb eps))) (* K5 (cos (lamb eps)))
                (* K6 (sin (lamb eps))))))
          (Qyz
           (lambda (eps)
             (- (* K2 (cos (lamb eps))) (* K1 (sin (lamb eps))))))
          (Qyeps
           (lambda (eps)
             (- (* K4 (cos (lamb eps))) (* K3 (sin (lamb eps))))))
          (Qy
           (lambda (eps)
             (+ (- (* rp1 (sin (lamb eps))))
                (- (* K5 (sin (lamb eps))))
                (* K6 (cos (lamb eps))))))
          (X1
           (lambda (eps Z)
             (+ (* (Qxz eps) Z) (* (Qxeps eps) eps) (Qx eps))))
          (Y1
           (lambda (eps Z)
             (+ (* (Qyz eps) Z) (* (Qyeps eps) eps) (Qy eps))))

; Point de l'helicoide
          (point-dev
           (lambda (eps Z)
             (vector (X1 eps Z) (Y1 eps Z) Z 1)))
```

```
        (point-dev-sym
         (lambda (eps Z)
           (SGvecpro (vector (X1 eps Z) (Y1 eps Z) Z 1)
                     (SGmatsmz (plan_sym Z)))))
;
;    ---- DEFINITION DES QUADRIQUES ENGLOBANTES ----------------
;
        (mmx-trapeze
         (lambda (eps Z)
           (list
            (point-dev eps Z)
            (vector
             (SGvecpro (SGx__med
                        (point-dev eps Z)
                        (point-dev (+ eps paseps) Z))
                       (SGmatmul
                        (SGmattrl (vector 0 0 pasZ -2))
                        (SGmatort (plan_sym Z))))
             (X_psi Z) (Y_psi Z) OZ)
            )))

        (mmx-galette
         (lambda (Z)
           (list
            (point-dev (epsf Z) Z)
            (vector
             (SGvecpro (SGx__med
                        (point-dev (epsf Z) Z)
                        (point-dev (epsa Z) Z))
                       (SGmatmul
                        (SGmattrl (vector 0 0 pasZ -2))
                        (SGmatort (plan_sym Z))))
             (X_psi Z) (Y_psi Z) OZ)
            )))

        (mmx-dent
         (let*
             (
              (face1 (point-dev (epsf Zmax) Zmax))
              (face2 (point-dev (epsa Zmax) Zmax))
              (face3 (point-dev-sym (epsf Zmax) Zmax))
              (face4 (point-dev-sym (epsa Zmax) Zmax))
              (dos1 (point-dev (epsf (+ pasZ Zmin)) Zmin))
              (dos2 (point-dev (epsa (+ pasZ Zmin)) Zmin))
              (dos3 (point-dev-sym (epsf (+ pasZ Zmin)) Zmin))
              (dos4 (point-dev-sym (epsa (+ pasZ Zmin)) Zmin))
              (extrem (list face1 face2 face3 face4
                            dos1 dos2 dos3 dos4))
```

```
               (Xmin (apply min (Vx extrem)))
               (Xmax (apply max (Vx extrem)))
               (Ymin (apply min (Vy extrem)))
               (Ymax (apply max (Vy extrem)))
               )
             (list
              (vector Xmax Ymax Zmax 1)
              (vector
               (vector (+ Xmin Xmax) (+ Ymin Ymax) (+ Zmax Zmin) 2)
                       OX OY OZ))
           ))
;
;    ---- DEFINITION DES VOLUMES -----------------------------
;
         (develop-trapeze
          (lambda (Z)
            (let repet ((trapeze_list (list))
                        (i 0))
              (if (< i inceps)
                  (repet
                   (cons
                    (min-max
                     (mmx-trapeze (+ (epsf Z) (* i paseps)) Z)
                     (DLuni
                      (color (vector (/ (- 6 i) 6) (/ i 6)
                                     (/ i 6) 1)
                             (trapeze
                              (point-dev (+ (epsf Z)
                                            (* i paseps)) Z)
                              (point-dev (+ (epsf Z)
                                            (* (+ i 1) paseps)) Z)
                              (vector 0 0 (- Z pasZ) 1)
                              (plan_sym Z)))
; L'ajout de la ligne ci-dessous permet de tracer les hexaedres de
; construction des differents volumes
;;                    (apply EXmechxa (mmx-trapeze
                                       (+ (epsf Z) (* i paseps))
                                       Z))
                      ))
                   trapeze_list)
                  (+ i 1))
                  (DLuni
                   (min-max (mmx-galette Z)
                            (apply DLuni trapeze_list))
;;                 (apply EXmechxa (mmx-galette Z))
                   ))
               )))

         (dent
```

```
          (let boucle ((liste-galette (list))
                       (i  incZ))
            (if (> i (- incZ))
                (boucle
                 (cons
                  (develop-trapeze (* i pasZ))
                  liste-galette)
                 (- i 1))
                (min-max
                 mmx-dent
                 (DLuni
                  (apply DLuni liste-galette)
;;                (apply EXmechxa mmx-dent)
                  ))
               )))
          )
;
;    ---- CORPS DE LA FONCTION DENT ----------------------------
;
        (DLuni
         dent
         (color gris-jaune
                (DLint
                 (visu_plan (plan_sym 25) .01)
                 (GDparseg (vector 0 0 25 1) O)
                 (GDqdrcyl (vector (+ ra1 1) 1) O OZ)))
         (color jaune
                (DLint
                 (visu_plan (plan_sym 0) .01)
                 (GDparseg (vector 0 0 -25 1) O)
                 (GDqdrcyl (vector (+ ra1 1) 1) O OZ)))
         )
        )))
```

A présent, penchons-nous sur le code de l'approximation quadratique de la denture (par une quadrique), le fichier source est le fichier "engrenage-quad.scm" (pour les différents points intervenant dans le code, le lecteur se reportera aux figures 5.5 et 5.6) :

```
;-------------------------------------------------------------
;   GENERATION DE LA DENTURE EN HELICE
;   Les parametres sont:
;     - m0 => module reel
;     - alpha0 => angle de pression reel (en degres)
;     - beta0 => angle d'inclinaison d'helice (en degres)
;     - bh => facteur multiplicatif pour la largeur de la
;                  denture (b = bh . m0)
;     - Z1 => nombre de dents souhaite sur l'engrenage 1
;     - Z1 => nombre de dents souhaite sur l'engrenage 2
;     - x1 => deport sur la denture 1
;     - x2 => deport sur la denture 2
;     - inceps => increment sur le parametre epsilon (eps)
;     - incZ => increment sur le parametre Z (extrusion)
;-------------------------------------------------------------

(define gear
  (lambda (m0 alpha0 beta0 bh Z1 Z2 x1 x2 inceps incZ)
    (let*
(
;
;    ---- FONCTION DIVERSES ----------------------------------
;
; idem que precedemment
;
;    ---- DEFINITION DES COULEURS ----------------------------
;
; idem
;
;    ---- DEFINITION DES FONCTIONS MATHEMATIQUES --------------
;
; idem
;
;    ---- DEFINITION DES PARAMETRES MECANIQUES (LAURIA) --------
;
; idem
;
;    ---- DEFINITION DES SOUS-FONCTIONS ----------------------
;
; idem
;
;    ---- DEFINITION DES QUADRIQUES ENGLOBANTES ---------------
;
; idem sauf que mmx-trapeze est inutile ici
```

```
;
;    ---- DEFINITION DES VOLUMES ------------------------------
;
; Fonction pour afficher le profil de l'hellicoide de Lauria
; (on trace des petites spheres sur chaque point du profil voulu)
        (develop-sphere
         (lambda (Z)
           (let repet ((point_list (list))
                       (i 0))
             (if (<= i inceps)
                 (repet
                  (cons
                   (GDsphdis
                    (vector 1 5)
                    (point-dev (+ (epsf Z) (* i paseps)) Z))
                    point_list)
                  (+ i 1))
                 (apply DLuni point_list))
            )))

        (dent-elementaire
         (lambda (Zini Zfin)
           (let*
               (
                (A1 (point-dev (epsf Zini) Zini))
                (A2 (point-dev (epsa Zini) Zini))
                (A3 (SGx___4x
                     A1
                     (tgtedevelop (epsf Zini) Zini)
                     A2
                     (tgtedevelop (epsa Zini) Zini)))
                (B1 (point-dev (epsf Zfin) Zfin))
                (B2 (point-dev (epsa Zfin) Zfin))
                (B3 (SGx___4x
                     B1
                     (tgtedevelop (epsf Zfin) Zfin)
                     B2
                     (tgtedevelop (epsa Zfin) Zfin)))

                (P0 A1)
                (P1 A2)
                (P2 A3)
                (P3 (if (= beta0 0)
                        OZ
                        (SGx__y_z
                         (SGx_zx3z A1 A2 B1)
                         (SGy__coo A3 B3))
                     ))
                (P4 (vector-ref (SGy2_cir (vector A1 A2 A3))
```

```
                                3))
 (P5 (vector-ref (SGy2_cir (vector B1 B2 B3))
                                3))
                 (quadric  (vector P0 P1 P2 P3 P4 P5))

                 (C1 (sym-point A1 Zini))
                 (C2 (sym-point A2 Zini))
                 (C3 (sym-point A3 Zini))

                 (D1 (sym-point B1 Zfin))
                 (D2 (sym-point B2 Zfin))
                 (D3 (sym-point B3 Zfin))

                 (Q0 C1)
                 (Q1 C2)
                 (Q2 C3)
                 (Q3 (if (= beta0 0)
                         OZ
                         (SGx__y_z
                          (SGx_zx3z C1 C2 D1)
                          (SGy__coo C3 D3))
                         ))
                 (Q4 (sym-point P4 Zini))
                 (Q5 (sym-point P5 Zfin))

                 (quadric2 (vector Q0 Q1 Q2 Q3 Q4 Q5))
             )
  ;    --------- Corps de la fonction dent-elementaire ---------
              (begin
                (DLuni
                 (min-max
                  (mmx-galette Zini)
                  (DLdif
                   (DLint
                    (DLformz2 quadric)
                    (DLformz2 quadric2)
                    (GDparseg
                     (vector 0 0 Zini 1)
                     (vector 0 0 Zfin 1))
                   (GDqdrcyl (vector ra1 1) O OZ))
                 (GDqdrcyl (vector ractif1 1) O OZ)))
; La ligne suivante permet de tracer les figures de constuction
; de la quadrique
;;               (EXmecqdr quadric)
;;               (apply EXmechxa (mmx-galette Zini))
               ))
             )))
  ;    --------- Fin du corps de la fonction dent-elementaire --
```

```
         (dent
          (let boucle ((liste-galette (list))
                       (i incZ))
            (if (> i (- incZ))
                (boucle
                 (cons
                  (DLuni
                   (color vert (dent-elementaire (* i pasZ)
                                                 (* (- i 1) pasZ)))
                   (color rouge (develop-sphere (* i pasZ)))
                   )
                  liste-galette)
                 (- i 1))
                (DLuni
                 (min-max mmx-dent (apply DLuni liste-galette))
;;               (apply EXmechxa mmx-dent)
                 )
             )))

         (ellipse
          (let*
              (
               (P0 (point-dev (epsf 0) 0))
               (P1 (point-dev (epsa 0) 0))
               (P2 (SGx___4x
                    (point-dev (epsf 0) 0)
                    (tgtedevelop (epsf 0) 0)
                    (point-dev (epsa 0) 0)
                    (tgtedevelop (epsa 0) 0)))
               (P3 OZ)
               (P4 (vector-ref (SGy2_cir (vector P0 P1 P2)) 3))
               (P5 (point-dev (epsf -5) -15))
               )
              (vector P0 P1 P2 P3 P4 P5)))

; Vecteur normal au point I
         (normale
          (SGvecpro (tgtedevelop epsI 0)
                    (SGmatrot (vector pi 2) (point-dev epsI 0) OZ)))

; Renvoie les 2 points d'intersection entre un rayon et une quadrique
         (intersec
          (SGqdrray
           (point-dev epsI 0) normale ellipse))

; Calcul de l'erreur (distance entre le point I de l'helicoide et le
; point d'intersection de la normale en ce point avec la quadrique
; d'approximation)
         (erreur
```

```
          (SGdisseg (point-dev epsI 0) (vector-ref intersec 0)))

      )
;
;    ---- CORPS DE LA FONCTION GEAR ----------------------------
;
; Pour afficher en mode texte certains resultats numeriques
     (begin
        (display rp1) (newline)
        (display ra1) (newline)
        (display intersec) (newline)
        (display (/ (vector-ref erreur 0) (vector-ref erreur 1)))
        (newline)

        dent
      )

    )))
```

Annexe H

Code de la bibliothèque de fonctions

Voici les principales fonctions utilisées dans notre bibliothèque de fonction. Elles sont lancées au démarrage de SGDLsoft, en incluant une ligne de chargement du fichier “library.scm” au sein du fichier “.init.scm”.

```
; Fonction de trace du repere, tres utile !!!

(define repere
  (lambda (r h)
    (let
        ((fleche
          (DLuni
           (GDcyl100
            (vector r 1)
            (vector (* 3 h) 4))
           (DLatt
            (SDmatrep
             (SGmattrl
              (vector  (* 3 h) 0  0 4)))
            (GDcon100
             (vector (* 2 r) 1)
             (vector h 4))))))
      (DLuni
       (GDsph000
        (vector r 1))
        fleche
       (DLatt
        (SDmatrep
         (SGmat001
          (vector  (SGcst_pi) 2)))
         fleche)
       (DLatt
        (SDmatrep
         (SGmat010
          (vector  (SGcst_pi) (- 2))))
         fleche)))))
```

```
; Fonctions de conversion deg->rad et rad->deg

(define rad
  (lambda (alpha)
    (/ (* (SGcst_pi) alpha) 180)))

(define deg
  (lambda (alpha)
    (/ (* alpha 180) (SGcst_pi))))

; Fonction carree

(define sqr
  (lambda (x)
    (* x x)))

; Fonction puissance

(define ^
  (lambda (x n)
    (if (= n 0)
        1
        (* x (^ x (- n 1))))))

; Fonction de visualisation de plans

(define visu_plan
  (lambda (vecteur epsilon)
    (let*
        ((vecteur (SGz__nor vecteur))
         (get (lambda (i) (vector-ref vecteur i)))
         (f1 (lambda (x) (+ (* -1 (get 3) x) (* -1 epsilon x))))
         (f2 (lambda (x) (+ (* -1 (get 3) x) (* epsilon x))))
         (A0 (vector (f1 (get 0)) (f1 (get 1)) (f1 (get 2)) 1))
         (A1 (vector (f2 (get 0)) (f2 (get 1)) (f2 (get 2)) 1))
         )
      (begin
        (set! valeur (list A0 A1))
        (GDparseg A0 A1)
         )
     )))

; Fonction de passage projectif->euclidien pour un point

(define voir
  (lambda (vecteur)
    (display (SGx__euc vecteur))
    (newline)))
```

Bibliographie

[1] David J. BUERGER
(1990) *LaTeX for Engineers & Scientists* , Collection “Computing that works”, New-York, McGraw-Hill Publishing Company Inc.

[2] Earle BUKINGHAM
(1949) *Analytical Mechanics of Gears* , 1re . éd., New-York, McGraw-Hill Publishing Company Inc.

[3] Jacques CHAZARAIN
(1996) *Programmer avec Scheme : de la Pratique à la Théorie* , Paris, International Thomson Publishing Inc.

[4] Steven K. FEINER , FOLEY J.D., HUGHES J.F. & VAN DAM A.
(1990) *Computer Graphics : Principles and Practice* , 2e . éd., Collection “Addison-Wesley Systems Programming Series”, Addison-Wesley Publishing Company Inc.

[5] Jérôme GERAERDS
(29/09/97) *Guide LaTeX*
http://www.admc.ulg.ac.be/private/∼Frederic/Latex/default.html

[6] Georges HENRIOT
(1979) *Traité Théorique et Pratique des Engrenages* , 6e . éd., p. 29-79 & 244-259, Paris, Dunod

[7] HUFFLEN J.-M. , ROEGEL D. & TOMBRE K.
(septembre 1998) *Guide local LaTeX du LORIA, Millésime 1998* , Version 2.14, http://www.loria.fr/tex/guide.html

[8] Toni JABBOUR
(1993) *Modélisation sur ordinateur des engrenages hélicoïdaux en plastique* , Thèse de doctorat, Génie mécanique, Ecole Polytechnique de Montréal

[9] Marie-Paule KLUTH
(28/02/1998) *FAQ LaTeX de l’équipe I&A, V. 2.12*
http://www.lri.fr/Francais/Recherche/ia/stuff/FAQ-LaTeX/

[10] Eitel H. LAURIA
(mai 1997) *Géométrie des engrenages parallèles extérieurs à denture hélicoïdale* , p. 77-113, Bulletin de l’I.E.T. (Institut de l’Engrenage et des Transmissions) 113

[11] Faydor L. LITVIN
(1994) *Gear Geometry and Applied Theory* , chap. 14-15, Englewood Cliffs - Prentice Hall

[12] Faydor L. LITVIN, M. HAWKINS, J. LU & D.P. TOWNSEND
(July 1997) *Computerized simulation of meshing conventional helical involute gears and modification of geometry* , NASA Technical Memorandum no 107451 & Army Research Laboratory Technical Report no ARL-TR-1370

[13] Keith RECKDAHL
(15/12/1997) *Using imported graphics in LaTeX 2ε*
Version 2.0, ftp://ftp.tex.ac.uk/tex-archive/epslatex.pdf

[14] Jean-François ROTGÉ
(1997) *L'aritmétique des formes : une introduction à la logique de l'espace* , Thèse de doctorat, Faculté de l'Aménagement, Université de Montréal

[15] Jean-François ROTGÉ & L. DANIEL
(1995) *Manuel technique de SGDLsoft (version 0.7)* , S.G.D.L. Systèmes Inc., Montréal

[16] Guy L.Jr. STEELE, G.J. SUSSMANN *et al.*
(1998) *Revised* ^{5} *report on the algorithmic language Scheme (R5RS)*
http://www-swiss.ai.mit.edu/scheme-home.html

[17] Lars VEDMAR
(1981) *On the Design of External Involute Helical Gears* , p. 4-24, Collection "Transactions of Machine Elements Division", Lund Technical University

[18] *(La)TeX navigator, Une encyclopédie(La)TeX*
(22/03/1999)
http://www.loria.fr/services/tex

Coordonnées des différents intervenants

Pour de plus amples renseignements, voici les coordonnées des auteurs :

– M. ARMENJON Frédéric
route de Soly
74250 FILLINGES
FRANCE
E-mail : farmenjon@hotmail.com

– M. GUENIFFEY David,
6, rue de Corcellot
21800 CHEVIGNY-SAINT-SAUVEUR
FRANCE
E-mail : dgueniffey@hotmail.com

De plus, si le lecteur souhaite entrer en contact avec nos différents partenaires, voici leurs coordonnées :

– M. BONIAU Henri
Directeur du Centre d'Enseignement et de Recherche de l'Ecole Nationale Supérieure d'Arts et Métiers de Cluny
C.E.R. E.N.S.A.M. Cluny
71250 CLUNY
FRANCE
Tél : (+ 011 33) 3-85-59-53-52

– M.OCTRUE
Directeur technique au Centre Technique des Industries Mécaniques
CETIM
52, avenue Félix-Louat - BP 80067
60304 Senlis Cedex
Tél. : (+011 33) 3-44-67-30-00
Fax : (+011 33) 3-44-67-34-00

– M. Rotgé Jean-François
SGDL Systèmes Inc.
400, avenue Laurier Ouest
Bureau 402
MONTREAL (QUEBEC), CANADA, H2V 2K7

Tél : (+ 001) (514) 948-0927
Site Internet : http://www.sgdl.com
E-mail : sgdl@sgdl.com, sgdl@videotron.ca

– M. TIDAFI Temy
Directeur du Groupe de Recherche en Conception Assistée par Ordinateur
G.R.C.A.O.
Faculté de l'Aménagement
Université de Montréal
2900, boul. Edouard-Montpetit
C.P. 6128, succursale Centre-Ville
MONTREAL (QUEBEC), CANADA, H3C 3J7
Bur: (+001) (514) 343-7923
Lab: (+001) (514) 343-2059
Télécopieur: (+001) (514) 343-2455
E-mail: Temy.Tidafi@umontreal.ca