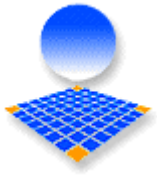
Ministère de l'Education Nationale

Université de Montpellier II

Rapport de projet informatique de l'Unité d'Enseignement GLIN405
de la Licence informatique 2ème année effectué
du 23/01/2012 au 15/05/2012

Modélisation d'objets 3D à partir d'un gabarit, rendu, éclairage et rotations

GAUTHIER Silvère
TAGHDA Samir



Remerciements

Nous tenons à remercier les personnes suivantes pour nous avoir soutenus :

MOUNTAZ Hascoët, pour nous avoir encadrés et dirigés dans la bonne direction, ainsi que pour nous avoir aidé à résoudre tous les problèmes directement liés au programme.

Ma petite amie (Silvère), pour m'avoir soutenu et motivé afin de réaliser ce projet dans de bonnes conditions mentales.

Certains élèves d'autres groupes de projet 3D pour avoir partagé leurs expériences avec nous, et pour l'entraide qui a pu exister entre nous.

Et pour finir, toutes les personnes ayant posté des tutoriaux sur les bibliothèques C, OpenGL et Glut, nous ayant permis d'acquérir les connaissances nécessaires à la réalisation de notre logiciel.

Table des Matières

1. Introduction

1.1 Généralités

- Objectifs :

Nous ne chercherons pas à révolutionner le monde du rendu 3D ni même de proposer un logiciel concurrent à d'autres déjà existants mais l'objectif majeur de ce projet est d'une part de nous familiariser avec les outils de programmation de rendu 3D et d'autre part d'approfondir nos connaissances en langage C. Pour cela, nous devrons apprendre également à nous auto-former et à gérer un groupe de travail, ceci nous préparant à l'avenir.

- Durée :

Ce projet se déroulera sur quatre mois avec pour finalité une soutenance oral permettant de présenter notre travail.

- Organisation :

Nous sommes quatre dans ce groupe de projet et allons nous répartir en deux binômes afin de travailler avec plus d'efficacité. La répartition des tâches se fera au sein de chaque binôme.

- Encadrement :

Des séances d'encadrement, en moyenne toutes les deux semaines, ont été prévues avec notre tuteur afin de nous aider grâce à des cours, nous donner des conseils personnalisés ainsi que de nous aider à corriger d'éventuels bugs que nous n'arriverions pas à déceler assez rapidement.

1.2 Le sujet

Le but de ce projet sera de modéliser des objets 3D à partir de gabarit.

Il s'agira donc dans un premier temps d'écrire le programme qui permette de dessiner le gabarit. Puis, dans un deuxième temps de construire l'objet 3D en construisant sa surface de révolution à partir du gabarit. Dans un troisième temps, il s'agira d'afficher et d'éclairer l'objet 3D ainsi obtenu. Enfin, il s'agira dans un quatrième temps de gérer les rotations de l'objet 3D selon un repère 3D qui pourra être placé soit au "centre" de l'objet soit en n'importe quel point arbitrairement choisi par l'utilisateur.

La bibliothèque utilisée pour effectuer ce projet sera OpenGL avec une programmation en C.

1.3 Cahier des charges

A) Introduction

1. Contexte

Ce projet est un projet scolaire n'ayant aucun but lucratif et n'ayant également aucun coût prévisionnel. De même les clauses juridiques et législatives, pouvant viser un projet tel que le

notre, ne seront pas prises en compte étant donné le but uniquement pédagogique de notre travail.

2. La modélisation 3D

Il existe une multitude de langage proposant des fonctionnalités 3D, mais il ne sont malheureusement pas tous portables, et pour éviter d'avoir à donner des fichier .dll ou à installer des bibliothèques supplémentaires sur les ordinateurs de la faculté, nous avons choisi d'utiliser Glut pour le fenêtrage et OpenGL pour le rendu 3D.

B) La demande

1. Description du projet

L'objectif du projet est de créer une petite application permettant de créer un objet en 3D à partir d'un gabarit créé par l'utilisateur. Celui-ci doit pouvoir modifier son gabarit et donc l'objet rendu, ainsi que faire tourner l'objet 3D selon l'axe choisi.

2. Fonctionnalités

L'application permettra à l'utilisateur de positionner quinze points dans la partie gauche de la fenêtre, qui se relieront alors dans l'ordre de création pour former un gabarit. Le premier et le dernier points se projetteront sur l'axe des ordonnées lors du rendu, dans le but d'avoir un objet 3D fermé.

L'utilisateur pourra ensuite déplacer ou supprimer les points qu'il désire, cela affectera également l'objet modélisé en temps réel. Il pourra également changer l'axe de rotation de l'objet (ou le remettre dans sa position initiale) grâce à des raccourcis clavier décrits dans le **Manuel d'utilisation**.

Certaines options seront ajoutées, telles que l'activation / désactivation des « wireframe » (fil de fer) ou du remplissage des polygones ou encore de l'éclairage, la modification de la précision horizontale de l'objet ainsi que sa vitesse de rotation.
Par la suite, pour une question de confort, les fonctionnalités de « undo » et « redo » seront ajoutées.

L'objet sera donc affiché en accord avec le gabarit dessiné, en rotation autour de l'axe par défaut ou, le cas échéant, celui défini par l'utilisateur, ainsi qu'avec toutes les options activées via les raccourcis clavier ou le menu.

En conclusion, notre projet permettra à un utilisateur de dessiner simplement un objet en 3D. Il pourra par la suite le modifier ainsi qu'activer ou désactiver les options implémentées.

C) Les contraintes

1. De coût

Aucunes, car l'accès aux documentations, aux logiciels, etc... est entièrement gratuit. Le développement sera effectué avec des logiciels gratuits.

2. De temps

Un délai est à respecter, le logiciel doit être rendu au plus tard la semaine n°20.

D) Déroulement du projet

1. Planification

Semaine 7 : réunion 1, rencontre des groupes et du tuteur, répartition des binômes.

Semaine 8 : recherches sur les fonctions du logiciel, apprentissage personnel sur Glut et OpenGL

Semaine 9 : réunion 2, élaboration du cahier des charges

Semaine 10 : réunion 3

Semaine 11 : début du développement

Semaine 12 : réunion 4

Semaine 13 : réunion 5

Semaine 14 : développement

Semaine 15 : développement

Semaine 16 : développement

Semaine 17 : réunion 6

Semaine 18 : réunion 7

Semaine 19 : préparation à la soutenance

Semaine 20 : soutenance orale

2. Ressources

Le partage des tâches étant difficile sur ce genre de projet, nous avons opté pour un travail en binôme et des mises en commun régulières au cours des réunions décrites ci-dessus.

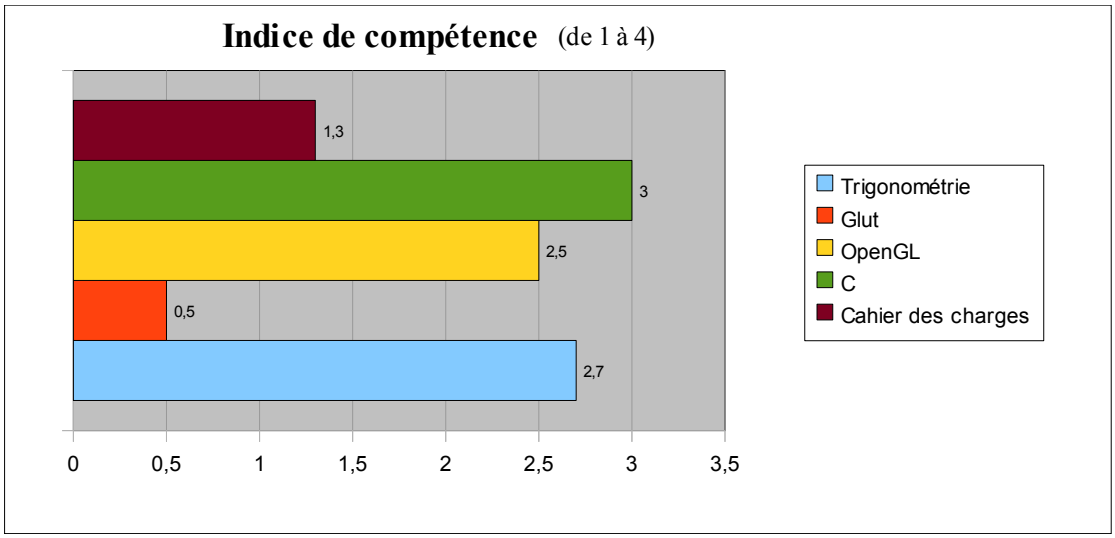
Liste des ressources humaines :

(1) Gauthier Silvère
(2) Taghda Samir

Conformément à l'énoncé présentant le projet, nous utiliserons les technologie, méthodes et langages suivants :

1. Les Méthodes de réalisation de cahier des charges.
2. Le langage C pour la réalisation du développement.
3. La bibliothèque OpenGL pour le rendu 3D de l'objet.
4. Les outils de Glut afin de concevoir l'interface graphique.
5. Les outils mathématiques permettant de calculer le modèle à rendre.

Sur le graphique suivant, on peut observer la synthèse du bilan préalable des connaissances et compétences de chacun des membres du projet sur les différents points énoncés ci-avant.



Les compétences et attentes des membres serviront à la répartition des binômes et à la distribution des tâches.

2. Organisation du projet

2.1 Organisation du travail

La première réunion nous a permis de créer deux binômes, à savoir Silvère / Samir et Adrien / Tristan.

La seconde nous a permis de commencer le travail et de savoir quoi utiliser pour réaliser le logiciel.

La troisième nous a familiarisé avec Glut et OpenGL afin de pouvoir développer chez soi.

Toutes les autres réunions nous ont permis de corriger des bugs ou de développer un peu en étant encadré, et donc de demander des conseils ainsi que des idées d'améliorations.

En conséquence d'un manque de présence, aucune réelle répartition des tâches n'a été faite et chacun a développé chez lui. Une mise en commun sera faite pour finaliser le projet et prendre le meilleur de chacun.

Etant alors en binômes, nous n'avons pas trouvé nécessaire d'élire un chef de projet.

Au niveau de la gestion du groupe, elle a été difficile car trop peu de contacts entre les membres se sont créés. Et l'absentéisme aux réunions n'a pas amélioré notre problème.

2.2 Choix des outils de développement

Pour la réalisation de ce projet, nous avons choisi plusieurs outils, à savoir :

1. Le langage C car c'est un langage de bas niveau dans lequel nous avons des connaissances approfondies. Il permet en outre d'utiliser un large choix de bibliothèques, graphiques ou non, étant essentielles au développement de notre application.

2. Le compilateur gcc car il est simple d'utilisation, déjà installé sur les machines possédant Unix, et parce qu'il permet de séparer compilation et édition de lien, de pré-compiler, etc... De plus, il est relativement puissant et permet de tout corriger grâce à l'option -Wall.

3. Les bibliothèques Glut pour le fenêtrage et OpenGL pour le rendu 3D, étant tous deux très portables et installés sur la plupart des machine sous Unix. De plus, ils sont relativement simples d'utilisation, de bas niveau et relativement puissants.

4. Le gestionnaire CodeBlocks pour la facilité de modularité du logiciel.

5. Les outils mathématiques de trigonométrie notamment, permettant ainsi de calculer le modèle à rendre. La bibliothèque math a donc été utilisée également.

3. Analyse du projet

Analyse préalable :

Le premier aspect de l'analyse a été de trouver comment séparer notre projet en différentes parties. Après réflexion, nous avons conclu que notre logiciel pouvait se découper en trois parties distinctes :

- La partie 2D où l'utilisateur pourra construire son gabarit, qui inclura donc les créations, modifications et suppressions des différents points du gabarit, ainsi que l'affichage à l'écran de ceux-ci.

- Le calcul du modèle 3D où chaque point de l'objet sera calculé et stocké dans des structures de données adaptées.

- Le rendu de l'objet, où le modèle sera affiché en faisant attention à l'ordre de lecture des points.

Analyse détaillée :

En détaillant l'analyse, le but était de choisir la manière d'implémenter, et donc les structures de données choisies, la façon de stocker les informations à l'intérieur de ces structures...etc.

1. Partie 2D : Gabarit

Pour une question de confort de programmation (accès aux différentes variables), nous avons choisi de créer une structure permettant de stocker les coordonnées du point créé, ainsi qu'un booléen permettant de savoir si le point est actif ou non (cela nous servira par la suite pour différentes options).

Nous avons ensuite choisi de créer un tableau de taille fixe contenant quinze fois cette structure, correspondant évidemment aux 15 points du gabarit, car le tableau en C est une structure contigüe et simple d'utilisation, surtout en mode statique.

Ainsi, nous stockerons les points créés dans l'ordre de création au sein de ce tableau, et pourrons simplement les afficher séquentiellement.

Au niveau du détail, nous choisirons un simple fond blanc, l'affichage de l'axe central de la figure ainsi que les points (grossis pour bien les voir) reliés entre eux dans l'ordre pour dessiner le gabarit.

2. Calcul du modèle 3D

Pour les même raisons que précédemment, nous avons opté pour un tableau en deux dimensions, permettant de stocker les structures correspondantes aux points, premièrement par « étage » (et donc par hauteur), et deuxièmement par « méridien » (au final, nous aurons construit l'objet à la manière d'un planisphère : par longitude et par latitude).

Un méridien sera donc simplement copié depuis le tableau du gabarit, et tous les autres seront calculés d'après celui-ci à l'aide des fonctions sinus et cosinus.

3. Rendu de l'objet en 3D

Pour le rendu, nous pensions simplement lire dans l'ordre de création, et donc dans l'ordre de stockage, en alternant entre deux étages pour permettre l'affichage des multiples faces de l'objet.

Nous avons, pour certaines options décrites dans le **Manuel d'utilisation**, dû séparer l'affichage des faces et des rebords de chaque face. En effet, nous devons faire deux rendus simultanés si les deux options sont actives. Nous nous sommes dit que ça ne changerait pas beaucoup les performances puisque notre application ne demandera théoriquement pas beaucoup de calculs.

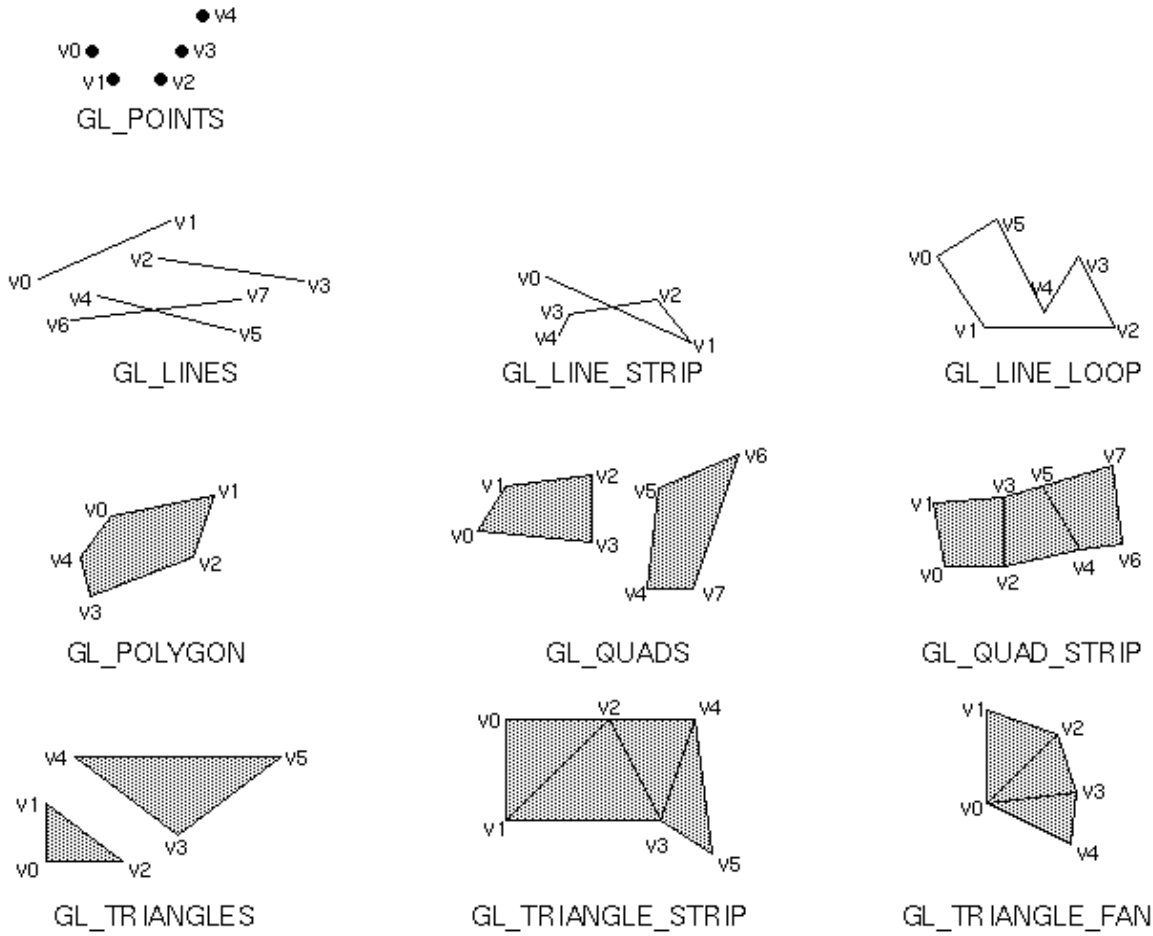
Pour le choix des modes de rendu OpenGL, nous avons choisi le mode GL_QUADS pour la plupart des faces, cela permettant d'avoir des faces rectangulaires. Nous avons aussi hésité avec des GL_TRIANGLES pour avoir un maillage plus précis, mais cet aspect ne nous intéresse pas pour ce projet.

Nous avons, par contre, choisi d'utiliser des GL_TRIANGLE_FAN pour les premier et dernier étage car cela nous permettait d'économiser le calcul de points inutiles étant positionnés au même endroit (je rappelle que le premier et le dernier point sont tous deux ramenés sur l'axe central, et donc ne changent pas lors du calcul du modèle).

Nous avons ensuite créé une vitesse de rotation affectant l'objet, en implémentant de manière à ce que le modèle ne soit pas calculé à chaque fois que l'objet tourne.

Les angles de rotation modifiables par l'utilisateur fonctionnent de la même façon.

Mémo des modes de OpenGL :



Les v_i sont les vertex (points) placés dans l'ordre de i croissant. Voilà l'affichage suivant le mode.

4. Développement 1

Partie 2D : Gabarit

Comme précisé précédemment, le gabarit est implémenté par un tableau de structures permettant le stockage des coordonnées de chaque point.

Premièrement, nous avons donc créé une structure stockant la coordonnée x (abscisse), la coordonnée y (ordonnée) et un booléen d'état « enable ».

Deuxièmement, nous avons initialisé un tableau de quinze points (structures) à l'origine et avec « enable » à 0.

Ensuite, nous avons du créer différentes fonctions telles que « ajouter un point », « supprimer un point », « déplacer un point »...
Ces différentes fonctions sont appelées lors de clic souris ou de touche de clavier, et donc sont appelées directement par Glut.

- Ajouter un point :

Cette fonction crée un point à l'emplacement du pointeur de la souris lorsque l'on actionne le clic gauche de celle-ci. Pour commencer, il faut vérifier au préalable si la souris est à l'intérieur de la partie « gabarit » de la fenêtre, si ce n'est pas le cas, rien ne sera modifié / créé.

Cette fonction doit donc parcourir séquentiellement le tableau du gabarit et regarder si chaque point est activé. Dès qu'un point inactivé est trouvé, son abscisse et son ordonnée sont affectées de celles de la souris (en accord avec la fenêtre bien sûr) et son booléen d'état est mis à « actif » (à 1).
S'il n'y a aucun point inactif, c'est que le gabarit est complet (dans la consigne, limité à quinze points) et la fonction n'ajoutera pas de nouveau point.

- Supprimer un point :

Ici, on cherche à supprimer le point pointé par la souris après clic droit de la souris.

Cette fonction doit donc récupérer les coordonnées du pointeur de la souris, pour ensuite parcourir le tableau gabarit à la recherche du point correspondant. À chaque étape, nous vérifions alors l'abscisse et l'ordonnée (seulement pour les points actifs bien sûr) et si elles correspondent à la position pointée (en réalité une fourchette correspondant à la taille visible du point), nous devons supprimer ce point.

Pour cela, il suffirait de mettre « enable » à 0. Mais le problème est que l'ordre des points risque de changer lorsque l'on ajoutera de nouveaux points par la suite. Pour résoudre cela, nous allons alors déplacer tous les points suivants d'un cran vers le début du tableau, ce qui gardera l'ordre de création des points et libèrera le dernier.

- Déplacer un point :

Ici, on veut pouvoir changer les coordonnées d'un point grâce à un glisser-déposer de la souris, fait par clic gauche.

Pour cela, nous devons créer une fonction récupérant les coordonnées du pointeur souris, cherchant ensuite le point correspondant, et lors d'un cliqué-glissé, changer en temps réel les coordonnées du point choisi.

Il nous a été difficile d'implémenter cela car nous devions alors créer des fonctions intermédiaires :

- une permettant de savoir si le bouton gauche de la souris était enfoncé (en réalité un simple booléen)
- une autre indiquant si le pointeur souris est sur un point (également utilisée pour supprimer)

Pour finir, il faut, lors d'un déplacement souris, testé si le bouton est enfoncé, tester si le pointeur est sur un point, et ensuite si ces deux conditions sont réunies, modifier les coordonnées du point en question. Cela est possible car la deuxième sous-fonction renvoie l'indice du tableau gabarit correspondant au point.

Quatrièmement, il ne nous restait plus qu'à faire un rendu, qui sera détaillé dans la partie **Développement 3**.

5. Développement 2

Calcul du Modèle

Comme précisé précédemment, le modèle est stocké dans un tableau à deux dimensions, l'une correspondant à la « longitude » et l'autre à la « latitude ». Tout cela se fera à l'intérieur d'une unique fonction.

Premièrement, nous avons donc créé ce tableau, qui peut alors contenir le nombre de point, multiplié par la précision horizontale maximale (une option permet de la modifier, elle n'est donc pas fixe) et donc est aussi un tableau statique et contigu. Ce tableau est initialisé de la même manière que pour le gabarit.

Deuxièmement, nous avons copié le tableau gabarit à l'intérieur, à la même « longitude » : 0.

Troisièmement, il fallait compléter le tableau. Puisque nous avions opté pour le mode GL_TRIANGLE_FAN, nous n'avions pas besoin de compléter ni la première « longitude », ni la dernière. Donc, il suffit alors, pour les autres longitudes, de calculer les « latitudes » manquantes.

Mais pour cela, il est plus pratique de d'abord connaître le nombre de points afin de savoir directement lesquelles compléter. Nous avons donc créé une petite fonction simple qui incrémente un compteur si le point est activé en parcourant séquentiellement le tableau gabarit.

Donc, chaque « latitude » est calculée en fonction de la précédente, c'est-à dire que si l'on a par exemple le tableau T[15][36], toutes les cases T[i][0] (avec i compris entre 0 et 14) sont copiées du gabarit, et les autres, à savoir les cases T[i][j] (avec i compris entre 0 et 14 et j entre 1 et 35), sont calculées grâce au cosinus et sinus des coordonnées des cases T[i][j-1].

La rotation se faisant pour nous autour de l'axe y, voici la matrice de rotation correspondante :

$$R_{\mathbf{y}}(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

Il nous suffisait alors juste de multiplier cette matrice avec le vecteur coordonnées de chaque point.

Nous avons également ajouté un calcul de la normale pour chaque point, permettant ainsi un meilleur rendu. Pour éviter la refonte de tout le code et des structures utilisées, ces normales seront malheureusement pas stockées mais calculées à chaque itération.

6. Développement 3

Rendu

Pour le rendu, il nous a suffit de parcourir le tableau modèle dans le bon ordre afin de créer des faces et les afficher.

Premièrement, nous avons construit le premier « chapeau » avec le mode GL_TRIANGLE_FAN. Pour cela, nous avons d'abord défini le tout premier point créé (ici T[0] [0]). Et, toujours avec ce mode, nous avons créé une boucle itérative définissant tous les points du deuxième « étage » (ici T[1][i]).

Deuxièmement, nous devions faire attention à l'ordre de définition des points, à savoir que GL_QUADS peut créer une sort de sablier au lieu d'un rectangle si l'on croise les points...

Nous devions donc procéder comme suit :
- définir le premier point du deuxième étage,
- définir le premier point du troisième étage,
- définir le deuxième point du troisième étage,
- définir le deuxième point du deuxième étage.

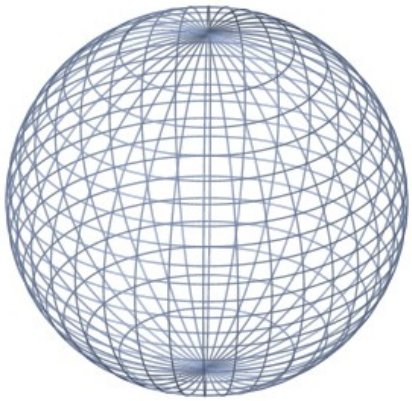
Formellement, l'ordre de définition est le suivant : T[i][j], T[i+1][j], T[i+1][j+1], T[i][j+1].
Cela crée toutes les faces rectangulaires entre deux étages, formant alors une ceinture autour de l'objet.

Il nous suffit alors de boucler itérativement sur ce procédé, pour créer les ceintures de l'objet, allant du deuxième étage à l'avant dernier.

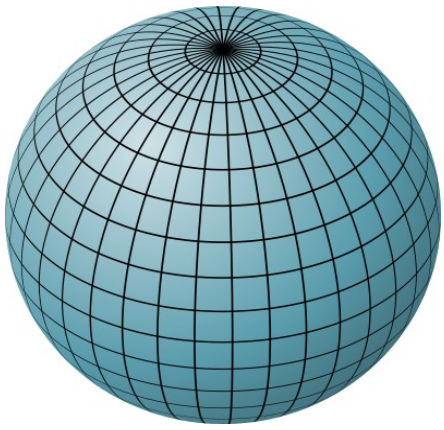
Troisièmement, nous avons construit le second « chapeau ». Nous avons donc procédé comme pour le premier, mais en prenant comme premier point (point fixe) le dernier point du modèle, et comme étage l'avant dernier.

Pour finir, nous avons refait tout le rendu mais avec le mode GL_LINE_STRIP pour obtenir un rendu en fil de fer.

Mémo des options d'affichage :



Mode « wireframe » ou fil de fer



Mode wireframe + filled (remplissage)

7. Manuel d'utilisation

L'utilisateur doit créer un gabarit sur la moitié gauche de la fenêtre. Pour cela, voici les manipulations possibles :

- Un clic gauche dans la partie blanche de la fenêtre crée un point
- Un glisser-déposé d'un point affiché le déplace
- Un clic droit sur un point affiché le supprime

L'utilisateur doit ainsi positionner des points qui se lieront alors **dans l'ordre de création** de ceux-ci. Cela formera un objet en trois dimensions qui sera affiché en rotation sur l'axe des ordonnées dans la partie droite.

Menu :

Un menu s'ouvre en cliquant sur le bouton de la molette de la souris. Toutes les options décrites dans les raccourcis clavier ci-dessous y sont présentes.

Raccourcis clavier :

- z : efface le dernier point créé
- y : ré-affiche le dernier point effacé
- c : supprime tous les point
- + : augmente la précision horizontale de l'objet
- − : diminue la précision horizontale de l'objet
- ← : change l'axe de rotation vers la gauche
- ↑ : change l'axe de rotation vers le haut
- → : change l'axe de rotation vers la droite
- ↓ : change l'axe de rotation vers le bas
- r : remet l'objet dans son axe de rotation initial
- PgUp : (Page Up) augmenter la vitesse de rotation de l'objet
- PgDn : (Page Down) diminuer la vitesse de rotation de l'objet
- p : arrête la rotation de l'objet
- w : active / désactive les fils de fer (wireframe)
- f : active / désactive le remplissage (filled)
- l : active / désactive la lumière
- Esc : ferme le programme

8. Perspectives et conclusions

8.1 Perspectives

Pour une suite future, notre application pourrait éventuellement sauvegarder et / ou charger des objets créés, mais cela obligerait à faire appel au système d'exploitation pour rechercher le fichier contenant l'objet (sans cela, nous pouvons créer un unique fichier mais cela n'aurait pas grand intérêt).

Elle pourrait également augmenter la précision verticale de l'objet, mais cela obligerait à utiliser des courbes entre les points, mais cela permettrait à l'avenir de lisser l'objet si la précision est assez grande.

Nous pourrions également ajouter des fonctionnalités comme la modification directe de l'objet 3D, tel un logiciel d'infographie 3D par exemple. Il est dommage que nous n'ayons pas encore le niveau pour cela, mais de nombreuses améliorations seraient possibles comme créer des palettes de couleur, créer des brush... etc.

Une autre petite idée plus simple à réaliser serait d'inclure un « color-picker » à l'intérieur de la fenêtre qui permettrait à l'utilisateur de choisir la couleur de l'objet, du fond, de la lumière...

Mais toutes ces idées nécessiteraient alors l'implémentation de Menus directement dans la fenêtre, et nous n'avons pas connaissance de cette fonctionnalité avec Glut.

8.2 Conclusions

8.2.1 Fonctionnement de l'application

Les bugs relevés sont :

- après création des deux premiers points, le rendu du segment de l'objet 3D n'est pas de la bonne longueur, et il semble y avoir une longueur minimale... Je n'ai trouvé aucune raison valable à cela.

- suite à ce premier bug, lors de la suppression d'un point d'un gabarit à trois points empêche le « rabattement » sur l'axe des ordonnées d'un des deux points restants... (il reste également ce segment bogué. Aucune solution trouvée.

- les faces entre le dernier et le premier méridien, uniquement lorsque l'utilisateur augmente au maximum la précision, se retrouvent en maillage plus élevé que les autres. Ceci serait corrigé si l'on définissait une structure face plutôt que point.

- enfin, le dernier bug relevé est la rotation non voulue de la lumière avec l'objet, ainsi que le rendu irrégulier de celle-ci au delà de trois points. Cela doit sûrement être dû au calcul des normales, ce qui serait résolu en faisant une refonte du code et en définissant une structure pour chaque face plutôt que pour chaque point.

Tout le reste fonctionne :

- Un point est bien créé si l'on clique dans la partie adéquate, en dehors d'un point existant et dans la limite de points autorisés.

- Ces points créés sont bien reliés, la suppression d'un point est fonctionnelle.
- Toutes les options fonctionnent.
- Le menu fonctionne également.

Les outils ont bien été choisis, mais l'analyse n'a peut-être pas été assez poussée car nous aurions dû définir les facettes plutôt que les points, cela aurait facilité les calculs ainsi que la complexité du code.

En conclusion, le logiciel répond à la demande et seule une option n'est pas tout-à-fait au point. Pour un projet qui ne s'est pas déroulé comme prévu au niveau du groupe, je trouve qu'il est plutôt bien abouti, et que seul le temps pourrait manquer à ce programme.

8.2.2 Fonctionnement du groupe de travail

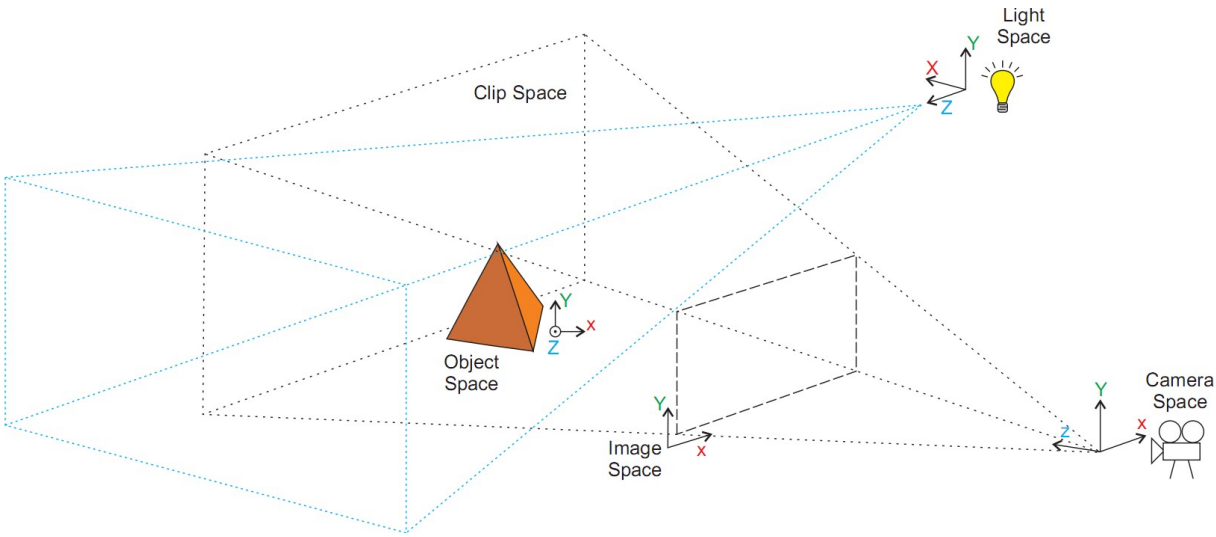
Malheureusement, nous n'avons pas réussi à « motiver les troupes », et le travail de groupe a été plus ou moins absent. L'efficacité en a forcément été diminuée, de par le manque d'implication de certains membres du groupe et par l'hétérogénéité évidente entre nous. En effet, certains membres se sont démotivés car ils n'y arrivaient pas et le temps a donc été insuffisant pour une seule personne, afin d'implémenter toutes les options voulues.

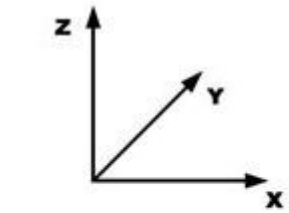
D'un autre côté, nous avons tous essayé de travailler sur tout le projet et donc ne nous sommes pas arrêté à notre partie, ainsi nous avons beaucoup plus appris que prévu. Le manque d'apprentissage au travail de groupe a donc été compensé par l'apprentissage de toutes les facettes de l'application.

En dehors de cela, le peu de contact entre les membres se sont déroulés harmonieusement et nous sommes tombés d'accord sur la plupart des aspects du projet. De plus, nous nous sommes entraidés non seulement à l'intérieur du groupe, mais également entre nous et d'autres membres d'autres groupes de projet 3D, ce qui a permis de résoudre quelques problèmes.

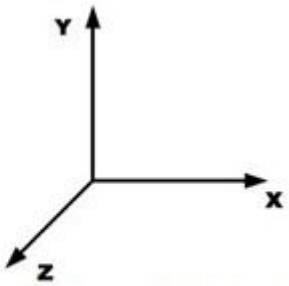
En conclusion, ce projet nous a appris énormément de chose, et surtout nous a appris à s'auto-former et à développer correctement sur un sujet au départ quasiment inconnu. Il nous prépare également à des projets futurs, que ce soit pour les études ou le professionnel.

9. Annexe A : documents d'analyse

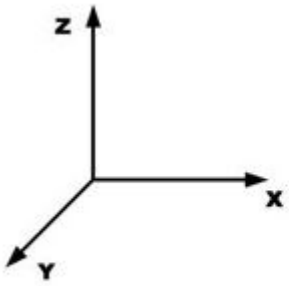




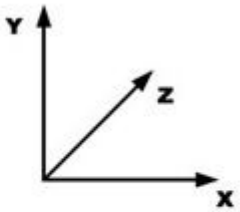
Right-handed, Z up



Right-handed, Y up



Left-handed, Z up



Left-handed, Y up

10. Annexe B : Listings identifiés et commentés

```
/*=========================================================================
Nom: main.c             auteur: Gauthier Silvère
Maj:  30/4/2012         Creation: 12/3/2012
Projet: Gabarit3D
-------------------------------------------------------------------------
Specification:
Ce fichier contient la boucle principale du programme.
=========================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h> /* inclut également gl.h et glu.h */

#include "def.h"
#include "var.h"
#include "fonc.h"
#include "glutfonc.h"
#include "dessin.h"
#include "model.h"
#include "rendu.h"

int main(int argc, char* argv[], char* env[]){

//Initialisation Glut et parametrage de la fenetre
  glutInit(&argc,argv);
  glutInitWindowPosition(0,0);
  glutInitWindowSize(SCREEN_WIDTH,SCREEN_HEIGHT);
  glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
  glutCreateWindow("Modelisation 3D depuis un gabarit");

  InitGL();  //paramétrage de OpenGL

//Fonctions de callback
  glutDisplayFunc(Draw);        //dessin
  glutVisibilityFunc(visible);  //visibilité : idle
  glutReshapeFunc(Reshape);     //parametrage de OpenGL selon la fenêtre
  glutKeyboardFunc(keydown);    //appuie sur une touche normale
  glutKeyboardUpFunc(keyup);    //relache la touche normale
  glutSpecialFunc(specdown);    //appuie sur une touche speciale
  glutSpecialUpFunc(specup);    //relache la touche speciale
  glutMouseFunc(mousedown);     //appuie ou relache un boutton souris
  glutMotionFunc(motion);       //cliquer glisser

//Menu
  int precMenu=glutCreateMenu(menu);
  glutAddMenuEntry("Increase (+)",220);
  glutAddMenuEntry("Decrease (-)",221);

  int vitMenu=glutCreateMenu(menu);
  glutAddMenuEntry("Increase (PgUp)",230);
  glutAddMenuEntry("Decrease (PgDn)",231);

  int viewMenu=glutCreateMenu(menu);
  glutAddMenuEntry("Wireframe (w)",20);
  glutAddMenuEntry("Fill (f)",21);
  glutAddMenuEntry("Light (l)",22);
  glutAddSubMenu("Precision",precMenu);
  glutAddSubMenu("Vitesse",vitMenu);

  int mainMenu=glutCreateMenu(menu);
  //glutAddMenuEntry("Load (à implémenter)",0);
  //glutAddMenuEntry("Save (à implémenter)",1);
```

```
  glutAddSubMenu("View",viewMenu);
  glutAddMenuEntry("Undo (z)",3);
  glutAddMenuEntry("Redo (y)",4);
  glutAddMenuEntry("Pause (p)",6);
  glutAddMenuEntry("Clear (c)",7);
  glutAddMenuEntry("Replace (r)",8);
  //glutAddMenuEntry("Aide (à implémenter)",9);
  glutAddMenuEntry("Close (Esc)",10);

  glutAttachMenu(GLUT_MIDDLE_BUTTON);

//Boucle d'évènements
  glutMainLoop();

return EXIT_SUCCESS;
}
```

```
/*=========================================================================
Nom: dessin.h           auteur: Gauthier Silvère
Maj:  11/4/2012         Creation: 12/3/2012
Projet: Gabarit3D
-------------------------------------------------------------------------
Specification:
Ce fichier contient les entêtes des fonctions nécessaires au rendu de la
fenêtre.
=========================================================================*/

#ifndef DESSIN_H
#define DESSIN_H


#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h> /* inclut également gl.h et glu.h */

#include "def.h"
#include "var.h"
#include "fonc.h"
#include "model.h"
#include "rendu.h"


void Draw();

#endif
```

```
/*=========================================================================
Nom: dessin.c           auteur: Gauthier Silvère
Maj:  30/4/2012         Creation: 12/3/2012
Projet: Gabarit3D
-------------------------------------------------------------------------
Specification:
Ce fichier contient les fonctions nécessaires au rendu de la fenêtre.
=========================================================================*/

#include "dessin.h"

void Draw(){
  glClearColor(255,255,255,0); //Fond blanc
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

//Axe y :
  glPushMatrix();
  glTranslated(SCREEN_WIDTH/4,0,0);
  glLineWidth(1);
  glBegin(GL_LINES);
    glColor3ub(0,0,0);
    for(int i=0;i<SCREEN_HEIGHT;i+=10)
      glVertex2d(0,i);
  glEnd();
  glPopMatrix();

//Trait central :
  glPushMatrix();
  glTranslated(SCREEN_WIDTH/2,0,0);
  glLineWidth(2);
  glBegin(GL_LINES);
    glColor3ub(0,0,0);
    glVertex2d(0,0);
    glVertex2d(0,SCREEN_HEIGHT);
  glEnd();
  glLineWidth(1);

//Fond bleu :
  glTranslated(1,0,-SCREEN_HEIGHT);
  glBegin(GL_QUADS);
    glColor3ub(0,0,0);
    glNormal3f(0.0,0.0,1.0);
    glVertex2d(0,0);
    glVertex2d((SCREEN_WIDTH/2)-1,0);
    glColor3ub(45,150,225);
    glVertex2d((SCREEN_WIDTH/2)-1,SCREEN_HEIGHT);
    glVertex2d(0,SCREEN_HEIGHT);
  glEnd();
  glPopMatrix();

//Points :
  glPushMatrix();
  for(int i=0;i<NB_PTS;i++)
    if(pts[i].enable==1){
      glBegin(GL_QUADS);
        glColor3ub(0,0,0);
glVertex2d(pts[i].x-2,pts[i].y-2);
glVertex2d(pts[i].x-2,pts[i].y+2);
glVertex2d(pts[i].x+2,pts[i].y+2);
glVertex2d(pts[i].x+2,pts[i].y-2);
      glEnd();
    }
  glPopMatrix();

//Segments reliant les points :
```

```
  glPushMatrix();
    glBegin(GL_LINE_STRIP);
      glColor3ub(0,0,0);
      for(int i=0;i<NB_PTS;i++)
        if(pts[i].enable==1)
          glVertex2d(pts[i].x,pts[i].y);
    glEnd();
  glPopMatrix();


//Figure 3D :
  Rendu3D(Modele3D(actPoint()));

  glFlush();

  glutSwapBuffers();
}
```

```
/*=========================================================================
Nom: fonc.h             auteur: Gauthier Silvère
Maj:  30/4/2012         Creation: 12/3/2012
Projet: Gabarit3D
-------------------------------------------------------------------------
Specification:
Ce fichier contient les entêtes des fonctions supplémentaires.
=========================================================================*/

#ifndef FONC_H
#define FONC_H


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <GL/glut.h> /* inclut également gl.h et glu.h */

#include "def.h"
#include "var.h"


// initialisation de OpenGL
void InitGL();

// fonction indiquant si le pointeur souris est sur un point
int onPoint(int x, int y);

// ajout d'un point s'il n'existe pas dans la limite de points construits
void addPoint(int x, int y);

// suppression d'un point
void delPoint(int i);

// récupération du nombre de points actifs
int actPoint();

// fonction appelée lors d'un choix dans le menu déroulant
void menu(int option);

// calcul de la normale pour chaque point
float* Normale(int i, int j);

#endif
```

```
/*=========================================================================
Nom: fonc.c             auteur: Gauthier Silvère
Maj:  30/4/2012         Creation: 12/3/2012
Projet: Gabarit3D
-------------------------------------------------------------------------
Specification:
Ce fichier contient les fonctions supplémentaires.
=========================================================================*/

#include "fonc.h"

void InitGL(){
  glEnable(GL_COLOR_MATERIAL);
  glEnable(GL_DEPTH_TEST);

  /*glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,MatSpec);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,MatDif);
  glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT,MatAmb);

  glLightfv(GL_LIGHT0, GL_DIFFUSE, Light1Dif);
  glLightfv(GL_LIGHT0, GL_SPECULAR, Light1Spec);
  glLightfv(GL_LIGHT0, GL_AMBIENT, Light1Amb);
  glLighti(GL_LIGHT0, GL_SPOT_CUTOFF, 90);*/

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glShadeModel(GL_SMOOTH);
}

int onPoint(int x, int y){
  for(int i=0;i<NB_PTS;i++)
    if(pts[i].x>x-5 && pts[i].x<x+5 && pts[i].y>y-5 && pts[i].y<y+5)
      return i;
  return -1;
}

void addPoint(int x, int y){
  int i=0;
  while(i<NB_PTS && pts[i].enable==1)
    i++;
  if(i<NB_PTS){
    pts[i].enable=1;
    pts[i].x=x;
    pts[i].y=y;
    undo=redo=0;
    }
}

void delPoint(int i){
  if(i<NB_PTS && i!=-1){
    while(i<NB_PTS-1){
      pts[i].enable=pts[i+1].enable;
      pts[i].x=pts[i+1].x;
      pts[i].y=pts[i+1].y;
      i++;
    }
    pts[i].enable=0;
    undo=redo=0;
  }
}

int actPoint(){
  int nb_pts=0;  //nombre de points créés
  for(int i=0;i<NB_PTS;i++)
    if(pts[i].enable==1)
```

```
      nb_pts++;
  return nb_pts;
}

void menu(int option){
  switch(option){
    case 0: break;

    case 1: break;

      case 20: wireframe=(wireframe==1? 0 : 1); break;
      case 21: fill=(fill==1? 0 : 1); break;
      case 22: light=(light==1? 0 : 1); if(light==0) glEnable(GL_LIGHTING);
 else glDisable(GL_LIGHTING); break;

        case 220: if(precHorizontal<PREC_MAX) precHorizontal+=13; break;
        case 221: if(precHorizontal>PREC_MIN) precHorizontal-=13; break;

        case 230: if(vitesse<VIT_MAX) vitesse+=0.1; break;
        case 231: if(vitesse>VIT_MIN) vitesse-=0.1; break;

    case 3: if(actPoint()>0){
     pts[actPoint()-1].enable=0;
     undo++;
     redo--;
      }
      break;

    case 4: if(redo<undo){
        pts[actPoint()].enable=1;
     undo--;
     redo++;
              }
      break;

    case 6: pause=(pause==1? 0 : 1); break;

    case 7: for(int i=0;i<NB_PTS;i++) pts[i].enable=0; break;

    case 8: angleX=angleZ=0.0; break;

    case 9: break;

    case 10: exit(EXIT_SUCCESS); break;
  }
}

float* Normale(int i, int j){  // 0 -> x ; 1 -> y ; 2 -> z
  float u1[3],u2[3],u3[3],u4[3];
  float n1[3],n2[3],n3[3],n4[3];

  if(i>0 && i<actPoint()-1){

    u1[0]=modele[i][j-1].x - modele[i][j].x ;
    u1[1]=modele[i][j-1].y - modele[i][j].y ;
    u1[2]=modele[i][j-1].z - modele[i][j].z ;

    u2[0]=modele[i+1][j].x - modele[i][j].x ;
    u2[1]=modele[i+1][j].y - modele[i][j].y ;
    u2[2]=modele[i+1][j].z - modele[i][j].z ;

    u3[0]=modele[i][j+1].x - modele[i][j].x ;
    u3[1]=modele[i][j+1].y - modele[i][j].y ;
    u3[2]=modele[i][j+1].z - modele[i][j].z ;
```

```
    u4[0]=modele[i-1][j].x - modele[i][j].x ;
    u4[1]=modele[i-1][j].y - modele[i][j].y ;
    u4[2]=modele[i-1][j].z - modele[i][j].z ;


    n1[0]=u4[1]*u1[2] - u4[2]*u1[1] ;
    n1[0]=u4[2]*u1[0] - u4[0]*u1[2] ;
    n1[0]=u4[0]*u1[1] - u4[1]*u1[0] ;

    n2[0]=u1[1]*u2[2] - u1[2]*u2[1] ;
    n2[0]=u1[2]*u2[0] - u1[0]*u2[2] ;
    n2[0]=u1[0]*u2[1] - u1[1]*u2[0] ;

    n3[0]=u2[1]*u3[2] - u2[2]*u3[1] ;
    n3[0]=u2[2]*u3[0] - u2[0]*u3[2] ;
    n3[0]=u2[0]*u3[1] - u2[1]*u3[0] ;

    n4[0]=u3[1]*u4[2] - u3[2]*u4[1] ;
    n4[0]=u3[2]*u4[0] - u3[0]*u4[2] ;
    n4[0]=u3[0]*u4[1] - u3[1]*u4[0] ;


    N[0]=n1[0] + n2[0] + n3[0] + n4[0] ;
    N[1]=n1[1] + n2[1] + n3[1] + n4[1] ;
    N[2]=n1[2] + n2[2] + n3[2] + n4[2] ;


    float l = sqrt( N[0]*N[0] + N[1]*N[1] + N[2]*N[2] );

    N[0] = N[0]/l;
    N[1] = N[1]/l;
    N[2] = N[2]/l;
  }
  return N;
}
```

```
/*==========================================================================
Nom: glutfonc.h         auteur: Gauthier Silvère
Maj:  11/4/2012         Creation: 12/3/2012
Projet: Gabarit3D
--------------------------------------------------------------------------
Specification:
Ce fichier contient les entêtes des fonctions associées à glut.
==========================================================================*/

#ifndef GLUTFONC_H
#define GLUTFONC_H


#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h> /* inclut également gl.h et glu.h */

#include "def.h"
#include "var.h"
#include "fonc.h"


// fonction appelée lorsque la fenêtre est minimisée ou rétablie
void visible(int vis);

// fonction appelée continuellement pour rafraîchir la fenêtre
void Idle();

// fonction appelée lors d'un redimensionnement de la fenêtre
void Reshape(int width, int height);

// fonction appelée lors d'un appui sur une touche clavier ASCII
void keydown(unsigned char key, int x, int y);

// fonction appelée lors d'un relâchement de touche clavier ASCII
void keyup(unsigned char key, int x, int y);

// fonction appelée lors d'un appui sur une touche clavier spéciale
void specdown(int key, int x, int y);

// fonction appelée lors d'un relâchement d'une touche clavier spéciale
void specup(int key, int x, int y);

// fonction appelée lors d'un appui sur un bouton de souris
void mousedown(int button, int state, int x, int y);

// fonction appelée lors d'un déplacement de souris
void motion(int x, int y);


#endif
```

```
/*=========================================================================
Nom: glutfonc.c          auteur: Gauthier Silvère
Maj:  30/4/2012          Creation: 12/3/2012
Projet: Gabarit3D
-------------------------------------------------------------------------
Specification:
Ce fichier contient les fonctions associées à glut.
=========================================================================*/

#include "glutfonc.h"

void visible(int vis){
  if (vis == GLUT_VISIBLE)
    glutIdleFunc(Idle);
  else
    glutIdleFunc(NULL);
}

void Idle(){
  int nWaitUntil = glutGet(GLUT_ELAPSED_TIME);
  int nTimer = glutGet(GLUT_ELAPSED_TIME);
  if(nTimer >= nWaitUntil){
    glutPostRedisplay();
    nWaitUntil = nTimer + (1000 / 20);
    if(pause==0)
      angleY=angleY+vitesse;
  }
}

void Reshape(int width, int height){
  glViewport(0,0,width,height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
glOrtho(0,SCREEN_WIDTH,0,SCREEN_HEIGHT,-(SCREEN_HEIGHT+1),SCREEN_HEIGHT+1);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  float pos[4]={3*SCREEN_WIDTH/4,3*SCREEN_HEIGHT/4,10,1};
  glLightfv(GL_LIGHT0, GL_POSITION, pos);
}

void keydown(unsigned char key, int x, int y){
  keys[key] = 1;
  switch(key){
    case 27: exit(EXIT_SUCCESS); break;  //echap

    case '+': if(precHorizontal<PREC_MAX) precHorizontal+=13; break;

    case '-': if(precHorizontal>PREC_MIN) precHorizontal-=13; break;

    case 'p': pause=(pause==1? 0 : 1); break;

    case 'w': wireframe=(wireframe==1? 0 : 1); break;

    case 'f': fill=(fill==1? 0 : 1); break;

     case 'l': light=(light==1? 0 : 1); if(light==0) glEnable(GL_LIGHTING);
else glDisable(GL_LIGHTING); break;

    case 'z': if(actPoint()>0){
     pts[actPoint()-1].enable=0;
     undo++;
     redo--;
      }
      break;
```

```
    case 'y': if(redo<undo){
        pts[actPoint()].enable=1;
     undo--;
     redo++;
             }
      break;

    case 'c': for(int i=0;i<NB_PTS;i++) pts[i].enable=0; break;

    case 'r': angleX=angleZ=0.0; break;
  }
}

void keyup(unsigned char key, int x, int y){
  keys[key] = 0;
}

void specdown(int key, int x, int y){
  skeys[key] = 1;
  switch(key){
    case GLUT_KEY_LEFT: angleZ--; break;

    case GLUT_KEY_RIGHT: angleZ++; break;

    case GLUT_KEY_UP: angleX--; break;

    case GLUT_KEY_DOWN: angleX++; break;

    case GLUT_KEY_PAGE_DOWN: if(vitesse>VIT_MIN) vitesse-=0.1; break;

    case GLUT_KEY_PAGE_UP: if(vitesse<VIT_MAX) vitesse+=0.1; break;
  }
}

void specup(int key, int x, int y){
  skeys[key] = 0;
}

void mousedown(int button, int state, int x, int y){
  switch(button){
    case GLUT_LEFT_BUTTON:
      switch(state){
case GLUT_DOWN:
  mb1d=1;
         if(x>0 && x<SCREEN_WIDTH/2 && y>0 && y<SCREEN_HEIGHT){
    int i=onPoint(x,SCREEN_HEIGHT-y);
    if(i==-1)
      addPoint(x,SCREEN_HEIGHT-y);  //repere souris != repere fenetre
    else
      indPoint=i;
  }
  break;
case GLUT_UP:
  mb1d=0;
  indPoint=-1;
  break;
      }
      break;
    case GLUT_RIGHT_BUTTON:
      switch(state){
case GLUT_DOWN:
  mb2d=1;
  int i=onPoint(x,SCREEN_HEIGHT-y);  //repere souris != repere fenetre
  if(i!=-1)
```

```
      delPoint(i);
  break;
case GLUT_UP:
  mb2d=0;
     }
  }
}

void motion(int x, int y){
  if(mb1d==1)
    if(x>0 && x<SCREEN_WIDTH/2 && y>0 && y<SCREEN_HEIGHT){
       if(indPoint!=-1){                                    // -> gérer les
collisions
        pts[indPoint].x=x;
        pts[indPoint].y=SCREEN_HEIGHT-y;
      }
    }
}
```

```
/*=========================================================================
Nom: model.h            auteur: Gauthier Silvère
Maj:  11/4/2012         Creation: 12/3/2012
Projet: Gabarit3D
-------------------------------------------------------------------------
Specification:
Ce fichier contient les entêtes des fonctions nécessaires au calcul du
modèle 3D.
=========================================================================*/

#ifndef MODEL_H
#define MODEL_H


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <GL/glut.h> /* inclut également gl.h et glu.h */

#include "def.h"
#include "var.h"


int Modele3D(int nb_pts);

#endif
```

```
/*============================================================================
Nom: model.c            auteur: Gauthier Silvère
Maj:  11/4/2012         Creation: 12/3/2012
Projet: Gabarit3D
----------------------------------------------------------------------------
Specification:
Ce fichier contient les fonctions nécessaires au calcul du modèle 3D.
============================================================================*/

#include "model.h"

int Modele3D(int nb_pts){
  if(nb_pts>1){
    double theta=2*PI/precHorizontal;
    modele[0][0].enable=1;
    modele[0][0].x=0;  //placement sur l'axe central du premier point
    modele[0][0].y=pts[0].y-SCREEN_HEIGHT/2;
    modele[0][0].z=pts[0].z;
    for(int i=1;i<nb_pts-1;i++){  //jusqu'à l'avant dernier point
      modele[i][0].enable=1;
      modele[i][0].x=pts[i].x-SCREEN_WIDTH/4;
      modele[i][0].y=pts[i].y-SCREEN_HEIGHT/2;
      modele[i][0].z=pts[i].z;
      for(int j=1;j<=precHorizontal;j++){
        modele[i][j].enable=1;
          modele[i][j].x=cos(theta)*modele[i][j-1].x + sin(theta)*modele[i]
[j-1].z;
        modele[i][j].y=modele[i][j-1].y;
          modele[i][j].z=cos(theta)*modele[i][j-1].z - sin(theta)*modele[i]
[j-1].x;
      }
    }
    modele[nb_pts-1][0].enable=1;
      modele[nb_pts-1][0].x=0;   //placement sur l'axe central du dernier
point
    modele[nb_pts-1][0].y=pts[nb_pts-1].y-SCREEN_HEIGHT/2;
    modele[nb_pts-1][0].z=pts[nb_pts-1].z;
  }
  return nb_pts;
}
```

```
/*=========================================================================
Nom: rendu.h            auteur: Gauthier Silvère
Maj:  11/4/2012         Creation: 12/3/2012
Projet: Gabarit3D
-------------------------------------------------------------------------
Specification:
Ce fichier contient les entêtes des fonctions nécessaires au rendu 3D de
l'objet.
=========================================================================*/

#ifndef RENDU_H
#define RENDU_H


#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h> /* inclut également gl.h et glu.h */

#include "def.h"
#include "var.h"


void Rendu3D(int nb_pts);

#endif
```

```
/*==========================================================================
Nom: rendu.c            auteur: Gauthier Silvère
Maj:  30/4/2012         Creation: 12/3/2012
Projet: Gabarit3D
--------------------------------------------------------------------------
Specification:
Ce fichier contient les fonctions nécessaires au rendu 3D de l'objet.
==========================================================================*/

#include "rendu.h"
#include "fonc.h"

void Rendu3D(int nb_pts){
  if(nb_pts>1){
    glPushMatrix();
      glTranslated(3*SCREEN_WIDTH/4,SCREEN_HEIGHT/2,0);
      glRotated(angleX,1,0,0);
      glRotated(angleY,0,1,0);
      glRotated(angleZ,0,0,1);

//Base initiale :
      if(fill){
        glBegin(GL_TRIANGLE_FAN);
          glColor3ub(220,115,10);
    glNormal3f(0.0,1.0,0.0);
                  glVertex3d(modele[0][0].x,modele[0][0].y,modele[0][0].z);
//point fixe
          for(int i=0;i<=precHorizontal;i++){
      glNormal3fv(Normale(1.0,(float)(i%precHorizontal)));
                       glVertex3d(modele[1][i%precHorizontal].x,modele[1][i
%precHorizontal].y,modele[1][i%precHorizontal].z);
          }
        glEnd();
      }
      if(wireframe){
        glBegin(GL_LINE_LOOP);
          glColor3ub(0,0,0);
          for(int i=0;i<precHorizontal;i++){
            glVertex3d(modele[0][0].x,modele[0][0].y,modele[0][0].z);
            glVertex3d(modele[1][i].x,modele[1][i].y,modele[1][i].z);
                     glVertex3d(modele[1][(i+1)%precHorizontal].x,modele[1]
[(i+1)%precHorizontal].y,modele[1][(i+1)%precHorizontal].z);
          }
        glEnd();
      }

//Milieu :
    for(int i=2;i<nb_pts-1;i++){
for(int j=0;j<=precHorizontal;j++){
  if(fill){
            glBegin(GL_QUADS);
              glColor3ub(220,115,10);
        glNormal3fv(Normale((float)i-1,(float)j));
      glVertex3d(modele[i-1][j].x,modele[i-1][j].y,modele[i-1][j].z);
        glNormal3fv(Normale((float)i,(float)j));
      glVertex3d(modele[i][j].x,modele[i][j].y,modele[i][j].z);
        glNormal3fv(Normale((float)i,(float)((j+1)%precHorizontal)));
                     glVertex3d(modele[i][(j+1)%precHorizontal].x,modele[i]
[(j+1)%precHorizontal].y,modele[i][(j+1)%precHorizontal].z);
        glNormal3fv(Normale((float)i-1,(float)((j+1)%precHorizontal)));
                  glVertex3d(modele[i-1][(j+1)%precHorizontal].x,modele[i-1]
[(j+1)%precHorizontal].y,modele[i-1][(j+1)%precHorizontal].z);
    glEnd();
  }
  if(wireframe){
```

```
    glBegin(GL_LINE_LOOP);
              glColor3ub(0,0,0);
      glVertex3d(modele[i-1][j].x,modele[i-1][j].y,modele[i-1][j].z);
      glVertex3d(modele[i][j].x,modele[i][j].y,modele[i][j].z);
                    glVertex3d(modele[i][(j+1)%precHorizontal].x,modele[i]
[(j+1)%precHorizontal].y,modele[i][(j+1)%precHorizontal].z);
                 glVertex3d(modele[i-1][(j+1)%precHorizontal].x,modele[i-1]
[(j+1)%precHorizontal].y,modele[i-1][(j+1)%precHorizontal].z);
    glEnd();
  }
}
    }

//Base finale :
    if(fill){
      glBegin(GL_TRIANGLE_FAN);
        glColor3ub(220,115,10);
  glNormal3f(0.0,-1.0,0.0);
                          glVertex3d(modele[nb_pts-1][0].x,modele[nb_pts-1]
[0].y,modele[nb_pts-1][0].z);  //point fixe
          for(int i=0;i<=precHorizontal;i++){
       glNormal3fv(Normale((float)nb_pts-2,(float)(i%precHorizontal)));
             glVertex3d(modele[nb_pts-2][i%precHorizontal].x,modele[nb_pts-
2][i%precHorizontal].y,modele[nb_pts-2][i%precHorizontal].z);
          }
      glEnd();
    }
    if(wireframe){
      glBegin(GL_LINE_LOOP);
        glColor3ub(0,0,0);
          for(int i=0;i<=precHorizontal;i++){
                          glVertex3d(modele[nb_pts-1][0].x,modele[nb_pts-1]
[0].y,modele[nb_pts-1][0].z);
                          glVertex3d(modele[nb_pts-2][i].x,modele[nb_pts-2]
[i].y,modele[nb_pts-2][i].z);
                                                glVertex3d(modele[nb_pts-2]
[(i+1)%precHorizontal].x,modele[nb_pts-2]
[(i+1)%precHorizontal].y,modele[nb_pts-2][(i+1)%precHorizontal].z);
          }
      glEnd();
    }
  glPopMatrix();
  }
}
```

```
/*=========================================================================
Nom: def.h              auteur: Gauthier Silvère
Maj:  11/4/2012         Creation: 22/3/2012
Projet: Gabarit3D
-------------------------------------------------------------------------
Specification:
Ce fichier contient les constantes de préprocesseur.
=========================================================================*/

#ifndef DEF_H
#define DEF_H


// définition de la constante PI car non trouvée dans math.h
#define PI (3.141592653589793)

// définition des dimensions de la fenêtre
#define SCREEN_WIDTH  800
#define SCREEN_HEIGHT  600

// définition des précisions horizontales minimale et maximale
#define PREC_MAX  49
#define PREC_MIN  10

// définition des vitesses de rotation minimale et maximale
#define VIT_MAX 0.35
#define VIT_MIN 0.05

// définition du nombre maximal de points du gabarit 2D
#define NB_PTS  15


#endif
```

```
/*=========================================================================
Nom: var.h              auteur: Gauthier Silvère
Maj:  16/4/2012         Creation: 22/3/2012
Projet: Gabarit3D
-------------------------------------------------------------------------
Specification:
Ce fichier contient les déclarations des variables globales du logiciel.
=========================================================================*/

#ifndef VAR_H
#define VAR_H


// déclaration du tableau de caractères ASCII
int keys[256];

// déclaration du tableau de caractères spéciaux
int skeys[256];

// déclaration de la structure de donnée permettant de stocker un point
typedef struct Point{ int enable; double x; double y; double z;}Point;

// déclaration du tableau Gabarit
extern Point pts[NB_PTS];

// déclaration du tableau Modèle
extern Point modele[NB_PTS][PREC_MAX];

// définition de la précision horizontale actuelle
extern int precHorizontal;

// définition des booléens utiles aux options
extern int pause, wireframe, fill, undo, redo, light;

// définition des angles de rotation courants
extern double angleX, angleY, angleZ;

// définition de la vitesse de rotation courante
extern double vitesse;

// déclaration de booléens indiquant l'état des boutons de souris
extern int mb1d, mb2d;

// déclaration d'un entier utile à plusieurs fonctions
extern int indPoint;

// déclaration du tableau de normale
float N[3];

#endif
```

```
/*==========================================================================
Nom: var.c              auteur: Gauthier Silvère
Maj:  16/4/2012         Creation: 22/3/2012
Projet: Gabarit3D
--------------------------------------------------------------------------
Specification:
Ce fichier contient les variables globales du logiciel.
==========================================================================*/

#include "def.h"
#include "var.h"

// initialisation du tableau Gabarit
Point pts[NB_PTS]={{0,0,0,0}};

// initialisation du tableau Modèle
Point modele[NB_PTS][PREC_MAX]={{{0,0,0,0}}};

int precHorizontal=PREC_MIN;

int pause=0, wireframe=1, fill=1, undo=0, redo=0, light=0;

double angleX=0.0, angleY=0.0, angleZ=0.0;

double vitesse=VIT_MIN;

int mb1d=0,mb2d=0;  //Mouse Button 1/2 Down

int indPoint=-1;  //Indice du point sélectionné
```