13/01/2020

Modélisation Géométrique et Synthèse d'images

Rapport de projet

Alexis Guyot – Issam Mokram – Sarah Belfadel – Thinhinane Saidi – Arnaud Francois – Tanguy Belicard

MASTER 1 INFORMATIQUE DIJON

Table des matières

I. Introduction

Notre groupe a travaillé sur le sujet numéro 12 de la liste, intitulé « Faites Deux Noëls ! ». Le but du projet était de modéliser une petite scène, où un cerceau se déplace sur une surface de Catmull-Rom bicubique. Au sommet de celui-ci se trouve un bilboquet, composé de parties en bois et d'une ficelle, qui se déplace en même temps et qui est animé comme si quelqu'un était en train de jouer avec (la boule s'élance et revient sur la pique, le fil ne se décroche pas). On souhaite enfin qu'une ambiance de Noël règne sur la scène, avec des couleurs vives qui clignotent.

En termes de concepts de synthèse d'images, ce projet est l'occasion d'en travailler un certain nombre, et même d'en découvrir quelques-uns. Parmi les connus que nous pourrons approfondir, on retrouve bien évidemment la modélisation géométrique et le placage de textures, deux concepts qui seront essentiels afin de générer les quelques objets nécessaires pour notre scène. La première difficulté pour cette partie réside dans la génération par facettes de la surface de Camtull-Rom, qui demande de bien comprendre en détails la méthode pour arriver à un tel résultat. La deuxième difficulté réside dans la création du fil du bilboquet, qu'on souhaite être une extrusion d'un cercle selon une courbe de Bézier. Le concept d'extrusion est un concept nouveau à appréhender, ce qui en fait une difficulté par rapport au reste. La deuxième grosse partie de ce projet concerne l'animation. Bien qu'une partie de cette dernière reste simple (rotation sur lui-même du cerceau, clignotement des lumières), une autre est bien plus complexe à mettre correctement en place. On peut citer parmi les difficultés de cette partie la trajectoire de la roue, qui devra suivre l'orientation de la pente. Cela demande d'être assez à l'aise avec les formules des courbes et des surfaces, de bien les comprendre, mais aussi avec les concepts de vecteurs normaux, tangents et bitangents, et des mathématiques cachées derrière, comme le calcul de ceux-ci à l'aide de dérivées et de produits vectoriels, les calculs d'angles entre vecteurs, etc. Enfin, la dernière partie du projet concerne les lumières en OpenGL. Les principales difficultés de cette partie seront de (re)découvrir comment fonctionnent les lumières en OpenGL, et de correctement calculer les normales lors des modélisations pour que cette dernière réagisse bien au contact de nos objets.

Le projet a été réalisé par un groupe de 6 étudiants, Alexis Guyot, Issam Mokram, Sarah Belfadel, Thinhinane Saidi, Arnaud Francois et Tanguy Belicard. Il est développé en C++ à l'aide des bibliothèques OpenGL, GLUT et Armadillo.

II. Développement du projet

A. Création des jouets

La première étape du développement de notre projet a consisté à modéliser les différents objets qui composeront par la suite notre scène. Dans un premier temps, intéressons-nous à la modélisation des différents jouets mentionnés par le sujet, à savoir le cerceau et le bilboquet.

1. Modélisation du cerceau

Notre cerceau est représenté sous la forme d'un tore. Nous avons fait une modélisation par facettes, pour laquelle nous avons utilisé un tableau pour stocker les sommets, et un autre pour les

faces. Pour l'afficher, nous avons utilisé GL_QUADS, qui dessine un quadrilatère à partir de quatre points, en reliant ces derniers dans le sens direct.

Pour rendre notre cerceau plus joli, nous l'avons colorié avec la fonction « glColor », de sorte que des bandes rouges et blanches (comme les sucres d'orge pour rester dans le thème) s'alternent sur chaque arc qui compose notre roue. Pour cela, rien de plus simple, si la face qu'on dessine se trouve sur un morceau pair, on dessine des points et des faces blanches, sinon on dessine en rouge.

2. Modélisation du bilboquet

Le bilboquet est en bois et est constitué d'un cylindre, d'une pointe et d'une sphère. La ficelle est une extrusion d'un cercle selon une courbe de Bézier.

Partie principale et boule

Pour réaliser le corps du bilboquet, on a besoin d'un cylindre qui représente le manche, d'un cône qui représente la pointe, et d'une sphère pour la boule. Nous avons appliqué une texture en bois sur l'ensembles des éléments. Ces derniers existent tous sous la forme de primitives de la bibliothèque GLUT, mais nous avons choisi d'utiliser nos propres fonctions, pour pouvoir y appliquer des textures à notre guise. Nous avons choisi une représentation par facettes, sans mémorisation des sommets, en dessinant avec les primitives d'OpenGL GL_TRIANGLE_FAN et GL_QUAD_STRIP. Nous avons choisi de ne pas mémoriser les coordonnées et les indices des faces, car nous utilisons déjà cette technique pour d'autres formes plus complexes (fil, surface), et que nous voulions éviter le risque de surcharger la mémoire vive avec notre programme. L'inconvénient de cette méthode est que OpenGL redessine à chaque rafraichissement de la scène les parties en bois du bilboquet. Cependant, puisque seules ces parties sont concernées, et qu'elles restent, après tout, des formes simples avec peu de sommets/faces, cet inconvénient n'est pas trop gênant d'un point de vue performances. Tous nos éléments sont centrés à l'origine et de rayon 1, pour pouvoir les déplacer facilement avec glTranslate. La hauteur du cylindre est de 2, tout comme celle du cône.

Pour le cylindre, on le discrétise en n méridiens. On a donc des facettes rectangulaires (faces latérales), et des facettes triangulaires (faces du haut et du bas du cylindre). Pour calculer les sommets du cylindre, on fait varier un angle entre 0 et 2π, afin de couvrir un cercle complet, et ce simultanément sur deux valeurs de l'axe Y, afin d'obtenir deux plateaux parallèles. L'angle varie à chaque fois d'un pas égal à 1 sur le nombre de sommets qui composent le cercle, et ce jusqu'à ce que le tour du cercle ait été fait. Puisque nous utilisons la primitive QUAD_STRIP, il faut calculer puis afficher successivement un sommet du plateau haut, et un du plateau bas. Pour dessiner les faces du dessus avec TRIANGLES_FAN, on commence par dessiner le centre du plateau, puis on tourne autour de ce dernier, toujours avec la même méthode de l'angle qui varie entre 0 et 2π.

Pour ajouter la texture bois sur les faces latérales, on « découpe » la texture selon le nombre de méridiens. Ainsi, plutôt que de mettre toute la texture sur chaque face du cylindre, on en met qu'un morceau, qui fait la même hauteur que l'image entière, mais qui ne prendre qu'une partie de sa largeur. Celle-ci s'obtient simplement en divisant la taille de la texture (soit 1) par le nombre de morceaux dont on a besoin, soit le nombre de faces. Pour les faces du haut, on utilise directement les coordonnées x et z, qui à cause des fonctions cos et sin qui dessinent le cercle, varient entre -1 et 1. On les convertit entre 0 et 1, et on va récupérer sur la texture les coordonnées obtenues. Pour que la lumière s'applique correctement sur le cylindre, on calcule les normales en chaque sommet. Celles

pour les sommets qui constituent les faces latérales suivent la direction du rayon depuis le centre du plateau. Celles pour les faces supérieures sont tout simplement des vecteurs dirigés dans le même sens que Y, et celles pour les faces inférieures la même chose mais dans le sens inverse.

Pour le cône, du point de vue du calcul des coordonnées des sommets, on fait quasiment la même chose, mais la face du haut devient un unique point [0,1,0]. Les faces latérales deviennent ainsi des faces triangulaires, mais à part ce léger changement le reste est similaire.

Pour la sphère, on utilise également sa représentation paramétrique. La normale à chaque sommet de la sphère correspond également à ses coordonnées, puisque le vecteur suit la direction commencée par le rayon. Pour sa texture, on utilise la formule dite « d'uv mapping de sphère », qui nous permet d'obtenir les formules suivantes (avec u et v les coordonnées dans l'espace de la texture qui correspondent à x, y et z dans l'espace de la scène) :

$$u = 0.5 + \frac{\arctan2(d_z, d_x)}{2\pi},$$
$$v = 0.5 - \frac{\arcsin(d_y)}{\pi}.$$

Source : https://en.wikipedia.org/wiki/UV_mapping

Maintenant que nous pouvons dessiner nos trois primitives, avec leur texture et leurs normales, on peut construire le bilboquet. Pour cela, on les génère une par une, puis on les déplace et les met à l'échelle correctement avec les fonctions glTranslate et glScale.

Ficelle

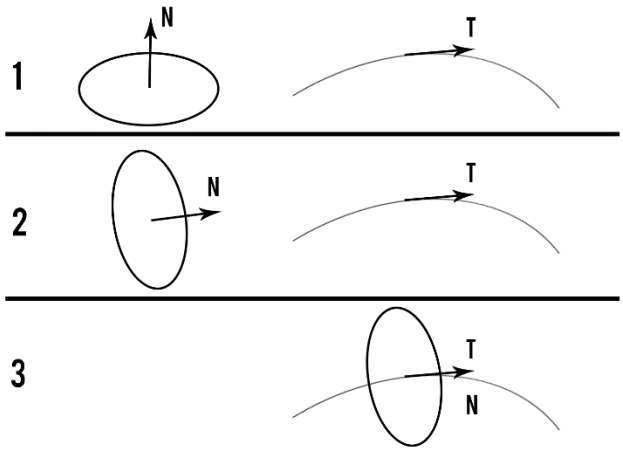
Pour représenter la ficelle de notre bilboquet, on souhaite construire son squelette à partir d'une courbe de Bézier, puis effectuer une extrusion d'un cercle selon ce dernier. Lorsqu'on parle d'extrusion, on sous-entend tracer un certain nombre de cercles, centrés sur un point de la courbe et dont l'orientation « suit le déplacement de cette dernière », puis les relier entre eux pour former une enveloppe autour de la courbe. Pour cela, on cherche à effectuer un ensemble de transformations sur la normale du cercle tracé (qui définit vers où est orientée la face du disque), afin de la faire correspondre avec la tangente de la courbe (qui définit vers où se dirige le tracé). L'intérêt d'une courbe de Bézier pour notre problème est que cette dernière interpole toujours le premier et le dernier point de contrôle du morceau tracé. Cela implique notamment qu'on peut ensuite très facilement placer notre fil sur notre scène, une fois celui-ci généré, puisqu'il suffit de donner les coordonnées du point où on souhaite que le fil commence, celles où il termine, puis placer les autres points en fonction de ces deux derniers pour obtenir la courbure voulue.

La première étape pour atteindre notre but est de calculer la courbure du fil (la courbe « squelette »). Pour cela, nous avons besoin de deux informations importantes : les points de contrôle qui définissent l'allure de la courbe, et la matrice de Bézier. La matrice de Bézier est connue, et est la suivante :

$$\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Nous reviendrons sur les coordonnées des points de contrôle dans la partie animation, puisque ces dernières sont amenées à changer dans le temps, pour réagir au déplacement de la boule. Ce qu'il faut juste savoir pour cette partie, c'est que nous avons besoin de 4 points de contrôle, pas forcément de plus puisqu'un seul morceau de courbe nous suffira pour notre fil. Les coordonnées de ces derniers fixées, il ne reste plus qu'à faire varier le paramètre « t » de la courbe entre 0 et 1, avec un pas égal à 1/npcm, npcm correspondant au nombre de disques qui composeront le squelette du fil. On peut alors calculer le centre de chaque cercle avec la formule « C(t) = [t^3 t^2 t 1] * Mbez * P », « t » étant toujours le même paramètre que nous faisons varier entre 0 et 1, Mbez étant la matrice de Bézier présentée ci-dessus, et P étant un vecteur-colonne constitué des 4 points de contrôle de notre fil.

Les futurs centres de nos cercles étant maintenant calculés, il reste à les orienter correctement afin qu'ils « fassent face à la direction dans laquelle continue la courbe de Bézier ». Cette notion un peu intuitive de « direction d'une courbe » se traduit en réalité par une tangente à cette dernière, et celle de « faire face » au fait de placer la normale du cercle sur celle-ci. Puisqu'une image vaut parfois mille mots, voici un petit schéma pour mieux visualiser l'effet recherché :



D'abord, avant de ramener quoi que ce soit sur une tangente de la courbe, encore faudrait-il pouvoir calculer cette dernière. Pour obtenir la tangente à une courbe, on la dérive selon son paramètre. Dans notre cas, cela correspond à dériver par rapport à la variable « t » qui parcourt la courbe. Seul le vecteur T^t se retrouve impacté, et devient [3t^2 2t 1 0]. Pour calculer la tangente en un point de notre courbe, on utilise donc la même formule que précédemment, mais en remplaçant le vecteur T par sa version dérivée. Ce calcul fait, le but est alors maintenant de faire en sorte que la normale de notre cercle (son axe Y) se retrouve avec la même orientation que la tangente. On va alors faire tourner notre plan, à l'aide de deux rotations. Premièrement, on va tourner autour de l'axe Y, afin que X ou Z se retrouve « juste en dessous de la tangente » (pour que cette dernière se retrouve dans le plan formé par les axes Y et X ou Z). On choisit que Z va former le plan avec Y. On calcule alors l'angle entre cet axe, dont les coordonnées sont (0,0,1), et le vecteur tangent, sans prendre en

considération sa coordonnée en Y, qui ne nous intéresse pas pour le moment. Pour obtenir la valeur d'un angle entre deux vecteurs non-nuls (ce qui est notre cas actuellement), on peut utiliser la formule suivante :

$$\cos(angle) = \frac{produit\_scalaire(vecteur1, vecteur2)}{norme(vecteur1) * norme(vecteur2)}$$

En plus du calcul du cosinus, on peut facilement obtenir celui du sinus, en prenant l'autre axe qui forme le plan dans lequel on calcule l'angle. L'intérêt de calculer les deux est d'ensuite pouvoir fabriquer nos propres matrices de rotations. On refait la même démarche pour la seconde rotation, qui a, elle, lieu autour de l'axe X. On peut alors générer nos matrices de rotation, qui sont de la forme :

$$R_y = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad et \quad R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & \sin\phi & 0 \\ 0 & -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Il ne reste ensuite plus qu'à appliquer ces dernières. Au niveau de l'algorithme, et pour être plus clair à ce niveau-là, ce dernier suit la logique suivante : Pour chaque morceau qui compose notre fil, on fait varier un paramètre t entre 0 et 1 avec un pas égal à 1 divisé par le nombre de cercles qui composent le squelette. Avec ce dernier, on calcule les coordonnées d'un point sur la courbe de Bézier, grâce à la formule de cette dernière. On calcule par la même occasion les coordonnées du vecteur tangent en ce point, avec la formule adéquate. Une fois cela fait, on génère nos matrices de rotation, avec la méthode expliquée précédemment, et on commence la génération du cercle en ce point de la courbe. Pour ce faire, pour chaque sommet qui compose la discrétisation du tuyau (nombre de sommets par cercle), on calcule sa coordonnée en utilisant la formule paramétrique du cercle (x = rayon * cos(alpha), y = rayon*cos(alpha), z=0 avec alpha qui varie entre 0 à 2π). Ces coordonnées maintenant obtenues, on les multiplie successivement par la matrice de rotation autour de Y, puis par celle autour de X. Il faut bien faire attention ici à l'ordre des matrices, la première qui s'applique étant celle la plus proche des coordonnées, donc celle directement à gauche. On additionne le tout avec les coordonnées du point sur la courbe pour effectuer une translation, et on stocke le résultat dans un tableau de sommets pour représenter la forme par facettes, et ne pas avoir à les recalculer à chaque fois. On remplit ensuite le tableau des faces, et enfin on affiche le tout, de la même façon que pour toutes les autres formes décrites dans le reste de ce rapport (avec glVertex, GL_QUADS, etc.).

C'est de cette façon que nous obtenons un fil généré selon une extrusion d'un cercle autour d'une courbe de Bézier. Comme dit précédemment, utiliser une courbe de Bézier nous permet de très facilement joindre deux objets avec le fil. Pour rattacher ce dernier avec le bilboquet, on place alors son premier point de contrôle sur une coordonnée appartenant à la sphère, et son dernier sur une coordonnée appartenant au cylindre. On peut ensuite déplacer les deux autres à notre guise pour modifier la courbure du fil.

B. Création de la surface

1. Génération et tracé de la surface

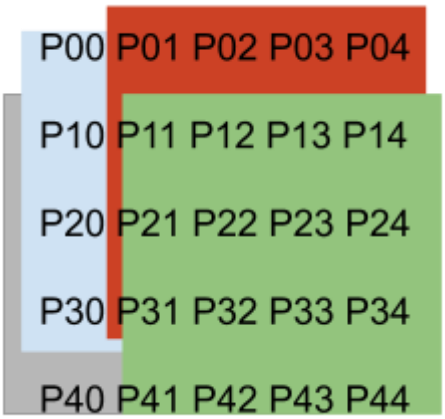
La surface à créer est une bicubique de Catmull-rom. L'objectif est d'obtenir une surface ondulée où on pourra déplacer notre tore. On peut considérer une surface comme la combinaison de deux courbes. La formule pour obtenir les coordonnées d'une surface est donc très proche de celle pour une courbe bicubique, à l'exception près qu'on fait cette fois-ci intervenir deux paramètres (u et v) qui varient tous les deux entre 0 et 1, plutôt qu'un seul dans le cas précédent (t). Les points de contrôle sont eux aussi un peu différents. En effet, puisque nous travaillons maintenant sur deux dimensions, un vecteur colonne de points de contrôle ne suffit plus. Il faut maintenant une matrice carrée, de dimension toujours égale à 4, puisque nous manipulons une surface bicubique (mais cette fois-ci donc 4x4 plutôt que 4x1). Il faut donc maintenant définir 16 points de contrôle pour tracer un morceau de surface, et non plus 4 comme nous faisions jusqu'ici pour une simple courbe. A part cela, on continue d'utiliser toujours la même matrice de Catmull-Rom, et on utilise toujours des vecteurs qui portent les paramètres de la surface, de la même manière que T ($[t^3\ t^2\ t\ 1]$) le faisait dans le cadre d'une courbe. Nous avons alors U et V, puisque nous avons deux paramètres « u » et « v ».

L'équation d'une surface de Catmull-Rom est la suivante :

$S(u,v) = [u^3\ u^2\ u\ 1]*M*G*M^T*[v^3\ v^2\ v\ 1]^T$

Avec M la matrice de Catmull-rom, G la matrice de points de contrôle, u et v les paramètres de notre surface et ^{T} indiquant qu'on travaille avec la transposée de cette matrice/de ce vecteur.

Dans notre cas M= $\begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$ avec la variable s égale à 0,5. Les points de contrôle de la matrice G sont générés afin d'assembler les morceaux de bicubique tel le schéma ci-dessous :



Chaque plan de couleur représente une matrice position G de dimension 4x4, qui servira pour dessiner un morceau de surface. Nous avons décidé d'utiliser 25 points de contrôle afin d'obtenir 4 morceaux différents (2 par axe).

Dans notre projet, une fonction spécifique appelée au lancement du programme permet de générer les points de contrôle. Ses coordonnées sont définies de la façon suivante :

- La coordonnée en X du point P_{ij} est la valeur de cos(PI*i/(N-1)).
- La coordonnée en Z du point P_{ij} est la valeur de 0.2*cos(PI*j/(N-1)).
- La coordonnée en Y du point P_{ij} est le résultat du calcul « cos(i)*sin(j) ».

Le choix des formules pour calculer les coordonnées des points s'est fait de façon relativement arbitraire, notre objectif étant juste à chaque fois de trouver une fonction qui, pour tout entier « x » en entrée, retournait une valeur comprise entre -1 et 1, afin de centrer notre surface vers l'origine avec deux morceaux de taille 1 sur chaque axe. Pour chaque morceau de courbe Mij, on sélectionne le point Pij et on construit la matrice des points de contrôle en prenant les trois points suivants pour chaque variable P(i+1)j, P(i+2)j, P(i+3)j, Pi(j+1), P(i+1)(j+1), …, P(i+3)(j+3). Une fois cela fait, on rajoute deux boucles imbriquées pour faire varier les paramètres u et v de la surface, entre 0 et 1 divisé par le nombre de sommets qui composeront notre morceau. On possède alors toutes les données nécessaires pour appliquer la formule ci-dessus, et récupérer les coordonnées en x, y et z de la surface en ce point, pour pouvoir le stocker dans un tableau. Nous pouvons également en profiter pour calculer la normale en ce point, pour que la lumière réagisse correctement à notre surface. Pour ce faire, on utilise la même formule que précédemment, mais à deux fois on la dérive selon un paramètre différent (comme pour le fil). La première fois on dérive en U, la seconde en V, et on obtient ainsi la tangente et la bitangente en ce point sur la surface. Il suffit alors de faire le produit vectoriel entre les deux pour obtenir notre normale, qu'on peut également stocker dans un tableau similaire à celui des coordonnées.

Comme pour le cerceau tout à l'heure, nous avons décidé d'utiliser une modélisation par facettes pour notre surface. Ainsi, nous n'aurons pas à recalculer les coordonnées de la surface à chaque passage dans la boucle principale, mais juste une fois au lancement du programme. Cette dernière s'obtient en trois temps : calcul des coordonnées des sommets, calcul des indices des faces de la forme, et enfin affichage. A chaque étape les calculs sont stockés dans des tableaux. Pour le deuxième calcul, celui des indices des faces, nous avons utilisé une technique similaire à ce que nous avons pris l'habitude de faire pour la plupart des formes à générer par facettes : Nous avons tracé un morceau de surface sur le papier, puis nous avons dressé la liste des sommets afin d'en extraire une formule générale pour la générer. Une fois ce petit exercice fait, pour chaque point de chaque morceau, on applique le résultat pour obtenir les quatre points qui constituent la face. A titre d'information, les formules extraites pour générer les indices de points de la surface sont les suivantes :

- Indice 1 : i * DiscSurf + j
- Indice 2 : i * DiscSurf + j + DiscSurf
- Indice 3 : i * DiscSurf + j + DiscSurf + 1
- Indice 4 : i * DiscSurf + j + 1

Avec i l'indice horizontal du point sur le morceau de surface, j l'indice vertical et DiscSurf le nombre de sommets sur chaque axe.

Pour en finir avec le dessin de la surface, on parcourt la liste des faces, et pour chaque indice on récupère ses coordonnées dans le tableau adéquat. On en profite également pour indiquer la normale en ce point, calculée précédemment, grâce à la fonction « glNormal3f » d'OpenGL, toujours pour que la lumière réagisse correctement. Et enfin, on en profite aussi pour plaquer correctement la texture, avec la méthode que nous allons rapidement expliquer dans la partie suivante.

2. Application d'une texture

L'application de la texture se fait, comme à l'accoutumée, juste avant le dessin du point lors de l'affichage de la surface. L'objectif dans cette partie est de paramétriser notre texture et notre surface, afin de les faire correspondre entre elles. Autrement dit, on souhaite « découper » l'image, de façon à pouvoir identifier des indices de sommets, dont nous connaissons déjà les coordonnées dans la scène, sur cette dernière. Pour cela, un découpage en deux temps est nécessaire.

D'abord, il faut séparer la texture pour que chaque morceau de la surface se voie attribuer un morceau de cette dernière. Notre surface est composée de deux morceaux verticalement (sur un plan 2D vu du dessus), et de deux morceaux horizontalement. De manière générale pour une Catmull-Rom bicubique, chaque axe possède N-3 morceaux, N étant la racine carrée du nombre de points de contrôle qui constituent la surface (indice maximal pour un point de contrôle = N-1, d'où N-3 et pas N-2).

Ensuite, il faut de nouveau séparer chaque « morceau de texture » obtenu, pour que chaque sommet du morceau de surface puisse s'identifier sur cette dernière. On découpe alors en un certain nombre de sous-parties, ce dernier correspondant à la discrétisation choisie précédemment pour constituer chaque morceau de la face.

Pour obtenir les coordonnées « u » et « v » sur la texture du point qu'on veut tracer, on commence par identifier l'indice du sommet par rapport à tous ceux qui composent la surface (tous les morceaux compris). Ce dernier est simplement l'addition de son indice sur son morceau de surface, avec le nombre de sommets avant lui sur l'axe (soit le numéro du morceau multiplié par le nombre de sommets par morceau). On divise ensuite le tout par le nombre de « morceaux sur la surface » obtenu après tous les découpages décrits précédemment, soit le nombre de faces par morceau, multiplié par le nombre de morceaux. On obtient la coordonnée en bas à gauche du carré qui constitue la face, et il suffit de faire le même calcul que précédemment mais avec l'indice par rapport à toutes les faces (numérateur) suivant, une fois uniquement sur l'axe horizontal, puis une fois uniquement sur l'axe vertical, puis sur les deux.

C. Création de l'ambiance lumineuse

1. Modélisation de la guirlande lumineuse

Pendant le développement de notre projet, les couleurs et les lumières de « Faites Deux Noëls » qui clignotaient nous ont fait penser à une guirlande lumineuse. Du coup, nous avons donc décidé d'en rajouter une à notre scène, même si les consignes ne l'exigeaient pas, afin d'améliorer le rendu et l'ambiance générale.

La guirlande est composée d'un fil, réalisé avec une extrusion d'une courbe de Bézier (voir partie « Modélisation du bilboquet -> Ficelle »), et de trois sphères qui représentent les lampes. Afin de

renforcer l'illusion que les ampoules soient à l'origine de la lumière de la scène, nous avons renforcé leur composante d'illumination grâce à la fonction OpenGL « glMaterial », accompagnée du paramètre « GL_EMISSION ». Cette composante définit l'intensité de la lumière émise par le matériau qui compose l'objet. Quand on invoque cette fonction avec une couleur particulière, une lumière de teinte équivalente semble être émise de notre forme, ce qui est tout à fait l'effet attendu pour nos lampes.

2. Illumination de la scène

Pour l'illumination de la scène, deux choix s'offraient à nous : soit nous utilisions les shaders avec GLSL, comme étudié ce semestre dans le module, soit nous utilisions les lumières natives présentes en OpenGL. Après concertation avec les membres du groupe, nous avons décidé d'utiliser plutôt la deuxième option, car nous ne nous sentions pas assez à l'aise avec les shaders dans la majorité, et que nous avions peur de perdre trop de temps en nous rajoutant une difficulté supplémentaire, alors que nous avions déjà assez à faire avec celles imposées par le sujet. De plus, utiliser les shaders nous aurait obligé à utiliser les VBO, ce qui aurait complexifié encore un peu le programme, et certains membres du groupe avaient encore plus de mal avec ce dernier concept qu'avec les shaders en eux-mêmes. Pour des raisons de simplicité, nous avons donc décidé de travailler avec les lumières natives d'OpenGL, ce qui, à défaut de plus nous faire manipuler certains concepts déjà travaillés et évalués cette année, nous aura permis d'en manipuler de relativement nouveaux.

On utilise une seule lampe OpenGL (GL_LIGHT). L'activation de cette dernière se fait en deux temps, d'abord en activant l'éclairage général avec la fonction glEnable(GL_LIGHTING), puis en allumant la lumière numéro 0 avec la fonction glEnable(GL_LIGHT0). Une lampe OpenGL est décrite par sa position (tableau de 4 éléments avec ses positions en x, y, z et le type d'éclairage de la lampe (0= directionnel 1= ponctuelle)), une description de sa composante spéculaire, une de sa diffuse et une de sa composante ambiante, afin d'appliquer le modèle de Phong. Une composante est décrite par un tableau de 4 éléments également, qui définissent sa couleur, les trois premiers étant les facteurs rouge, vert et bleu, et le dernier le facteur d'opacité « alpha ». On peut aussi définir un facteur d'atténuation pour l'éclairage de la lampe.

On a choisi pour notre scène un spot qui va changer de couleur pour simuler une guirlande lumineuse. On décrit donc les lumières avec des variables plutôt que des constantes, pour permettre les variations dans le temps.

Voici les options que nous avons définies pour notre éclairage :

- Position de la lampe : x = 0, y = 10, z = 10
- Composante diffuse : gris -> rouge = 0.5, vert = 0.5, bleu = 0.5
- Composante spéculaire : couleur changeante -> rouge = r, vert = g, bleu = b
- Composante ambiante : couleur changeante -> rouge = r-0.2, vert = g-0.2, bleu = b-0.2
- Facteur d'atténuation : x = 0.4, y= 0.4, z = 0.4

D. Animation de la scène

OpenGL permet l'animation de scènes grâce à sa fonction **glutIdleFunc**. Cette dernière, déclarée avant le lancement de la boucle principale de l'application, prend en paramètre une fonction, qui sera

appelée une fois à chaque passage dans cette dernière boucle. Nous allons nous en servir dans le cadre de ce projet afin de déplacer nos objets sur la scène et de modifier l'ambiance lumineuse.

1. Déplacement du cerceau

Génération de la trajectoire

Pour générer une trajectoire pour notre cerceau sur la surface de Catmull-Rom, le défi technique à relever est de réussir à tracer une courbe 2D qui suit et s'inscrit sur une surface 3D.

D'abord, la première étape a été de décider à quoi allait ressembler notre trajectoire. Nous aurions pu en effet choisir qu'il s'agirait d'un simple cercle, mais pour plus de flexibilité nous avons préféré une courbe de Catmull-Rom. Cette dernière a l'avantage d'interpoler ses points de contrôle, ce qui est pratique pour décider facilement l'allure de notre trajectoire. Comme déjà expliqué en détails à de nombreuses reprises dans ce rapport, on définit un certain nombre de points de contrôle, ici 7, afin de dessiner une courbe en N-3 morceaux (ici 4 donc). On dispose d'une variable appelée « posTraj », qui dans la fonction d'animation s'incrémente de 0 jusqu'à 100 multiplié par le nombre de morceaux de courbe, borne supérieure incluse. Afin de connaître la position sur la courbe où va se trouver notre tore, il faut d'abord savoir sur quel morceau de courbe on se trouve, et ensuite quelle est la valeur du paramètre de ce morceau de courbe. Pour cela, rien de plus simple, on divise la variable « posTraj » par le nombre de morceaux de courbe, le chiffre de la centaine nous indiquant le morceau, et on obtient la valeur de « t » en ramenant cette valeur entre 0 et 100 grâce à un modulo. Ces deux informations obtenues, on peut appliquer la formule de la courbe de Catmull-Rom en x puis en z pour obtenir les coordonnées du tore sur ces deux axes là. La prochaine étape est ensuite de faire correspondre ces coordonnées, qui sont dans l'espace de la scène, avec elles-mêmes mais cette fois-ci dans l'espace de la surface. De façon intuitive, nous allons maintenant essayer de « déposer » la courbe de trajectoire sur la surface, et observer où se situe le point identifié à cette étape une fois cette manipulation effectuée.

Puisque les choses sont bien faites, nous possédons actuellement deux valeurs, et la formule d'une surface demande deux paramètres. Nous allons alors tout simplement demander à la surface de calculer les coordonnées, dans son espace propre, d'un sommet qui aurait les coordonnées calculées précédemment dans l'espace de la scène, comme si vous allions « déposer le point au bon endroit ». On reprend alors la formule de la surface de Camtull-Rom vue précédemment, et on remplace « u » et « v » par la courbe-trajectoire en x et en z (résultats du paragraphe précédent). On a également besoin d'identifier les numéros des morceaux sur lesquels on se trouve verticalement et horizontalement, pour sélectionner les bons points de contrôle pour la surface. Pour ce faire, on commence d'abord par mettre nos deux espaces à la même dimension. En effet, notre courbe-trajectoire est tracée entre 0 et 1 sur les axes x et z. Nos deux variables u et v sont alors comprises entre ces deux bornes. On sait, grâce aux parties précédentes de ce rapport, qu'un morceau de notre surface est également compris entre 0 et 1. Cependant, il a également été précisé que cette dernière possédait plusieurs morceaux (N-3 pour être plus exact), et donc que de manière générale elle était dessinée non pas dans l'intervalle [0-1], mais de 0 à N-3. On effectue alors une mise à l'échelle, en multipliant nos variables u et v par le nombre de morceaux de notre surface. La partie entière des résultats précédents nous donne alors le numéro du morceau sur l'axe x pour u, et sur l'axe z pour v. On soustrait ces numéros aux résultats pour obtenir les coordonnées de u et de v, mais cette fois-ci non pas par rapport à toute la surface (espace de la scène), mais par rapport au morceau sur lequel elles se trouvent (espace de la surface).

Ensuite, en utilisant la formule de la surface vue précédemment, une fois en x, une fois en y, et une fois en z (en prenant la coordonnée ou bien en x, ou bien en y, ou bien en z des points de contrôle), on obtient finalement les coordonnées d'un nouveau point, qui correspond à l'endroit où dessiner le tore pour qu'il soit bien placé sur la trajectoire. Juste après le dessin de sa primitive, qui le centre par rapport à l'origine, on fait une translation aux coordonnées calculées par cette étape.

Orientation correcte par rapport à la surface

Une fois les étapes précédentes effectuées, notre cerceau est placé au bon endroit sur la surface, mais reste perpendiculaire au plan XZ (et peut donc rentrer dans la surface à certains endroits). On cherche alors maintenant à le faire tourner afin qu'il soit à la fois perpendiculaire à la surface, et orienté dans le sens de la trajectoire. On retrouve alors ici les mêmes problématiques que dans la partie sur le fil, où nous avons mis en place une stratégie afin de placer les axes du repère de notre objet de façon qu'ils correspondent à un ou plusieurs vecteurs prédéfinis (précédemment faire correspondre l'axe Y d'un cercle à la tangente d'une courbe de Bézier). Ici, on cherchera à faire correspondre le repère X, Y, Z du tore à celui formé par la normale à la surface et par la tangente et la bitangente de la courbe trajectoire sur la surface.

Pour la normale, on va la calculer à partir de la tangente et de la bitangente en ce point sur la surface (à ne pas confondre avec celles mentionnées juste avant, qui sont bien relatives à la courbe sur la surface). La méthode est toujours la même que quand nous avons calculé la normale en tout sommet de la surface pour l'éclairage. On calcule dans un premier temps la tangente à la surface, en dérivant la formule par rapport à u (U devient [3cx² 2cx 1 0], avec cx la coordonnée calculée à la première étape de la partie précédente). Puis on fait de même mais en dérivant cette fois-ci par rapport à v, ce qui nous donne la bitangente. On fait le produit vectoriel de ces deux vecteurs et on obtient la normale.

Pour calculer la tangente de la courbe sur la surface, on dérive la formule de la surface de Catmull-Rom. Puisque cette dernière est une fonction composée, qui prend en paramètre deux fonctions, la formule de sa dérivée est un peu particulière. Pour l'expliquer, on appelle « s » la formule de la surface, « cx » la formule de la courbe pour obtenir la coordonnée en x, et « cz » la formule de la courbe pour obtenir la coordonnée en z, « s'_u » la dérivée de la formule de la surface par rapport au paramètre u, et « s'_v » celle par rapport au paramètre v. La formule de la surface de Camtull-Rom est de la forme suivante : s(cx(t), tz(t)). Sa dérivée a pour formule : s'(cx(t), cz(t)) = s'_u(cx(t), xz(t)) * cx'(t) + s'_v(cx(t), cz(t)) * cz'(t). En appliquant correctement cette dernière formule, on obtient les coordonnées de la tangente en t de la courbe sur la surface de Catmull-Rom. On peut pour finir obtenir la bitangente en calculant le produit vectoriel entre la normale et la tangente.

Une fois les trois vecteurs précédents calculés, on possède un nouveau repère, qui se déplace de façon perpendiculaire à la surface, tout en suivant la trajectoire de la courbe. L'objectif maintenant est de fusionner ce dernier avec celui du cerceau, pour que ce dernier acquière ces mêmes propriétés. Pour ce faire, il suffit de trouver les transformations nécessaires pour passer de la matrice de départ (que nous appelons ici Md), la matrice identité, à la matrice que nous venons de calculer (que nous appelons ici Ma), composée de la fusion des trois axes Bitangente, Normale, Tangente et du vecteur translation (0, 0, 0, 1). Concrètement, on cherche alors la matrice Mt qui, multipliée par Md, donne Ma. On a alors Ma = Mt * Md, soit par simple inversion d'équation Mt = Ma * inverse(Md). Une fois la matrice de transformation calculée, on l'applique simplement à la matrice courante qui représente

notre scène (MODELVIEW), à l'aide de la fonction glMultMatrixf. Nos jouets suivent ainsi correctement la trajectoire définie sur la surface.

2. Animation du bilboquet

L'animation du bilboquet s'effectue en deux parties. D'abord, on cherche à modifier la position de la boule, afin que celle-ci donne l'impression d'être jetée depuis le manche. Pour cela, on utilise une simple variable qui s'incrémente et se décrémente jusqu'à deux valeurs limites, indéfiniment. Dès qu'une borne est atteinte, le sens d'incrémentation s'inverse (de 1 à -1 ou vice versa). La valeur de cette variable représente l'écart entre le manche et la boule, et la faire varier dans un sens ou dans un autre réduit ou agrandit l'espace. Pour le déplacement, on effectue une simple translation de la sphère sur l'axe Y, de la valeur de la variable.

Dans un second temps, on anime le fil afin que celui-ci reste correctement accroché et se tende et se détende en fonction de la hauteur à laquelle est la boule. Plus la boule est haute, moins le fil est courbé, donnant l'impression qu'il se tend, et inversement. Toute cette partie est basée sur les points de contrôle qui définissent l'allure de la courbe. Comme nous l'avons vu dans la partie sur la construction du fil, le premier et le dernier point de contrôle définissent ensemble où celui-ci commence et où il finit. Il suffit donc de prendre en compte l'écart défini par la variable ci-dessus pour faire monter le premier point de contrôle de la même façon qu'il fait monter la boule. Ce sont les deux points de contrôle du milieu qui définissent l'allure général du fil. Afin de donner une impression de pliage et de dépliage de ce dernier, nous avons utilisé la stratégie suivante : les deux points du milieu montent également en même temps que la boule, mais à une vitesse différente. En plus de cela, l'un des deux, celui le plus proche du manche se déplace également légèrement sur l'axe X, ce qui le fait s'approcher ou s'éloigner de ce dernier. Le résultat de cette manipulation est que la courbure est plus ou moins forte, et donc que le fil semble plus ou moins tiré.

3. Guirlande qui clignote

La troisième animation travaillée concerne l'ambiance lumineuse de notre scène. Comme nous l'avons expliqué dans une des sous-parties précédentes, nous avons construit une guirlande lumineuse pour habiller notre environnement. L'étape suivante était donc de simuler le clignotement de cette dernière, et de la synchroniser avec la lumière qui éclaire notre scène pour renforcer l'effet.

Comme pour les animations précédentes, c'est dans notre fonction « anim » que sont effectuées toutes les modifications nécessaires pour cette partie de l'animation. Nous avons vu dans la sous-partie précédente, dédiée à la lumière dans notre projet, que la couleur des différentes composantes de la lumière (seulement « diffuse » et « ambiante ») était décidée par la valeur de trois variables globales sobrement appelées « r », « g » et « b ». Le principe de notre animation consiste à faire varier ces trois valeurs, afin d'obtenir à chaque fois une couleur différente par combinaison. On a ainsi 4 couleurs, 'rose, 'rouge', 'vert' et 'jaune', qui s'alternent en boucle tous les 20 passages dans la fonction de la façon suivante :

De Zéro à 19 passages => rose.

De 20 à 39 => rouge.

De 40 à 59 => vert.

De 60 à 79 => jaune et on considère qu'on revient à zéro.

Ces mêmes variables servent également à définir la couleur émise par le matériau qui compose les lampes de la guirlande, ce qui donne l'impression que c'est le clignotement de ces dernières qui définit l'ambiance lumineuse générale de la scène.

4. Animations manuelles

Ces quelques animations sont un peu différentes des précédentes, puisqu'elles ne sont pas automatiques mais provoquées par l'utilisateur avec les touches du clavier.

D'abord, nous avons donné la possibilité de modifier rapidement la position de la caméra pour pouvoir passer sur plusieurs points de vue différents. Trois angles de vue différents sont proposés, et on accède à chacun en appuyant sur les touches 'v', 'b' et 'n' du clavier.

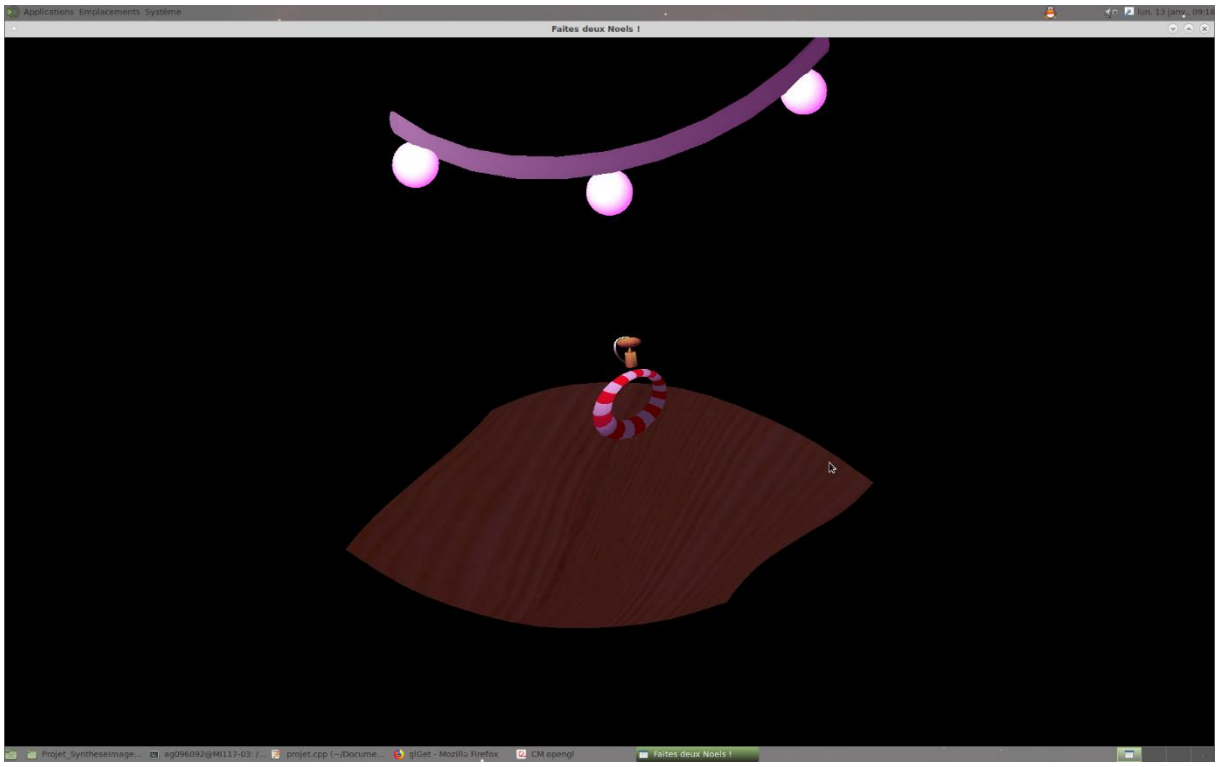
Ensuite, il est possible d'afficher la scène de trois façons différentes : de façon classique avec des faces pleines, et sous forme de fils de fer. Ces modes s'activent en appuyant respectivement sur les touches 'f' et 'p'. A noter que ces deux animations-là ont été récupérées des fichiers utilisés l'année dernière dans le module de Synthèse d'Images.

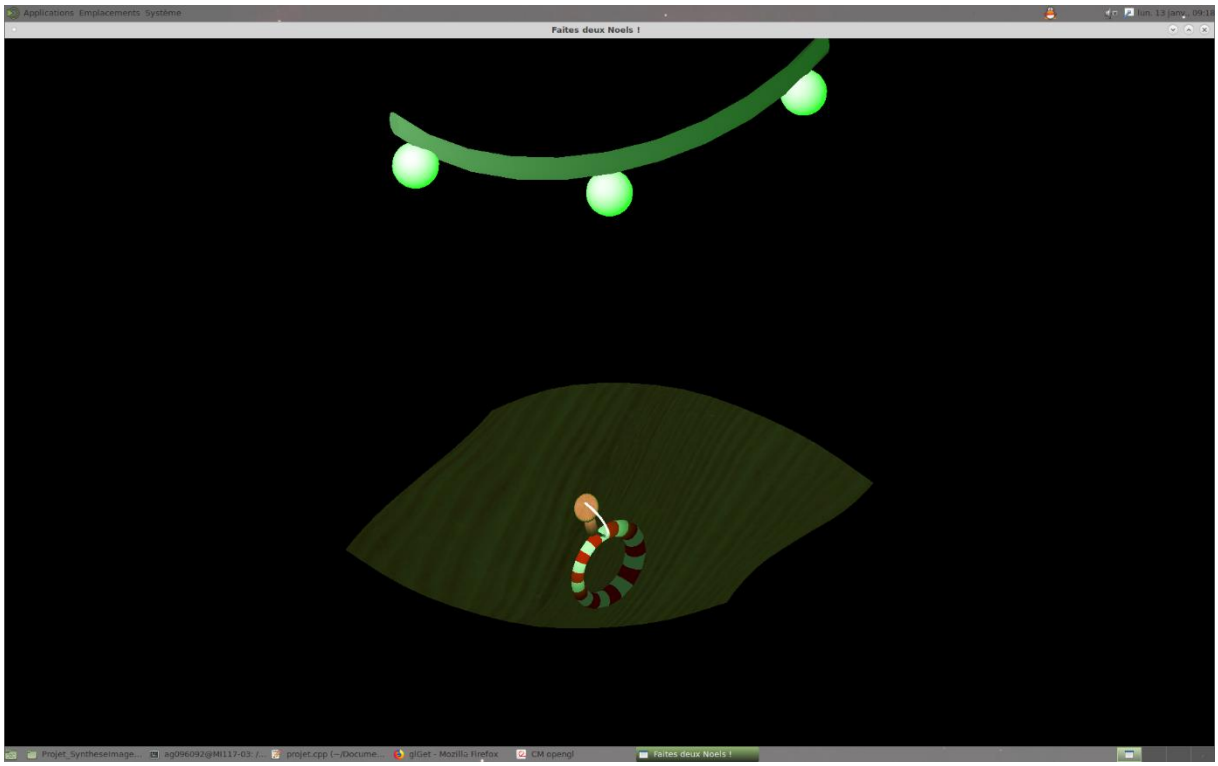
Enfin, l'utilisateur a la possibilité d'afficher la trajectoire que suit le cerceau sur la surface en appuyant sur la touche 't'.

III. Captures d'écran du projet

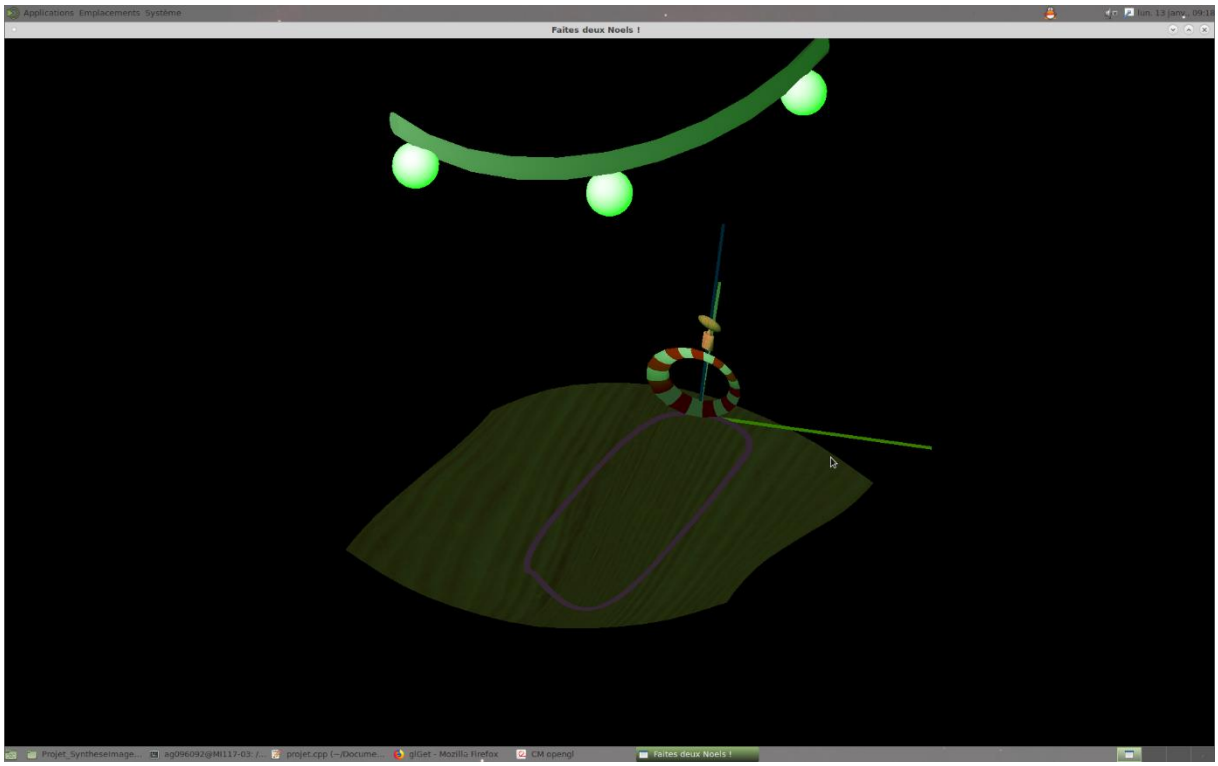
Voici, avant de terminer, quelques images du projet en action, pour rendre les explications données dans ce rapport plus concrètes.

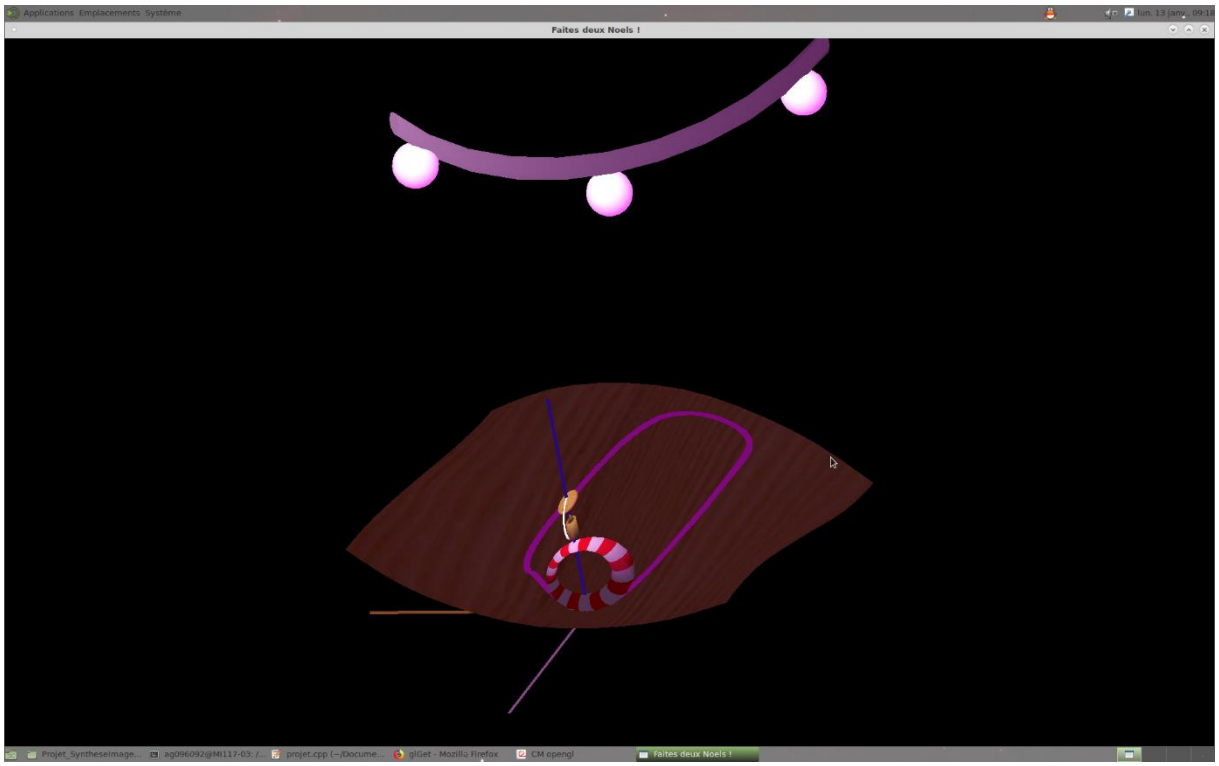
A. Sans affichage de la trajectoire

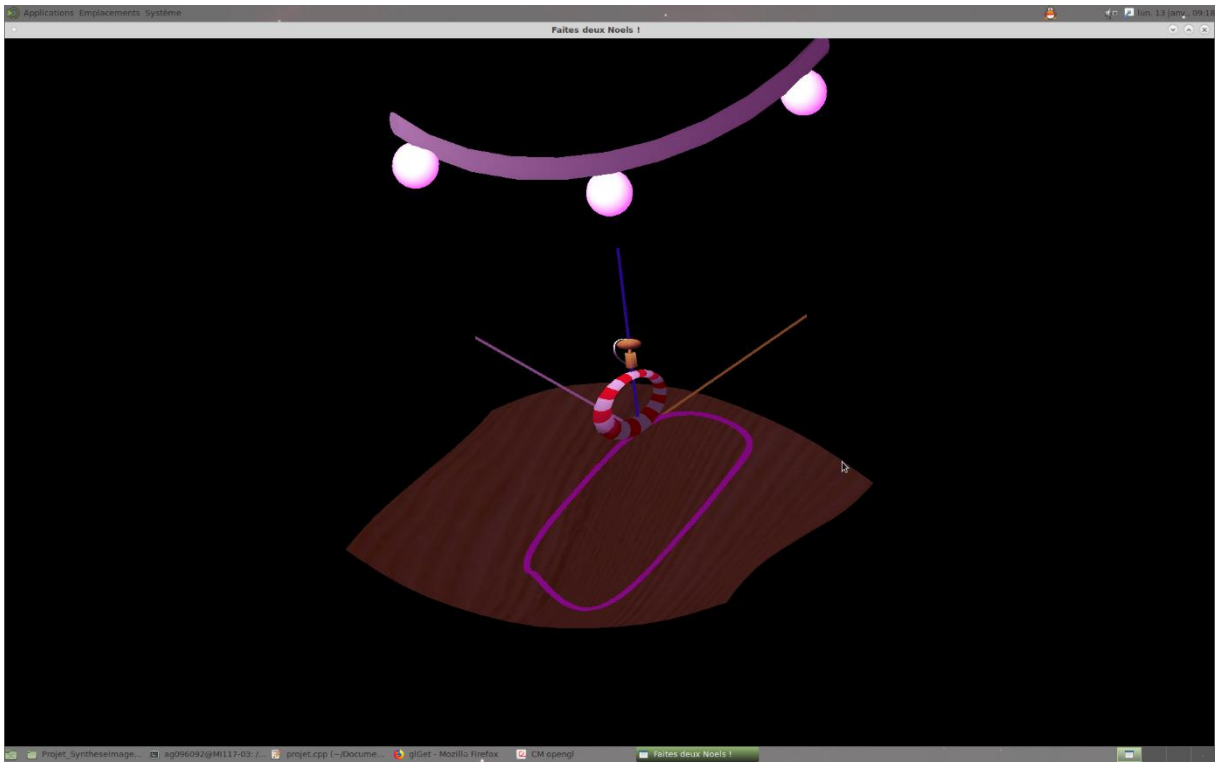




B. Avec affichage de la trajectoire







IV. Conclusion

En résumé, nous avons développé à travers ce projet une petite animation, aux couleurs de Noël, d'un cerceau et d'un bilboquet qui se déplacent sur une surface ondulée, représentée par une surface de Catmull-Rom.

Nous avons exploré et approfondi de nombreux concepts de synthèse d'images, comme la modélisation géométrique, les textures, la paramétrisation ou encore les lumières, mais le cœur de ce projet était réellement les courbes et les surfaces. Travailler sur ce sujet nous aura ainsi permis de mieux comprendre le fonctionnement des algorithmes, et de manière plus générale, des formules des courbes et des surfaces de Bézier et Camtull-Rom. Cela aura également été l'occasion de découvrir un nouveau concept, l'extrusion, qui permet, entre autres, de générer des tuyaux, des fils, ou n'importe quelle forme qui suit une courbe (en tout cas dans notre situation).

Les principales difficultés rencontrées pendant ce projet ont tourné autour des surfaces, surtout par rapport au déplacement du cerceau sur cette dernière. Il a fallu comprendre les formules dans un premier temps certes, mais en plus réussir à faire la différence entre les différents types de tangentes (à la surface, à la courbe sur la surface), sans se perdre entre tous les espaces dans lesquels nous avons travaillé (l'espace de la scène, de la surface, des textures, …). On est donc beaucoup plus sur des difficultés théoriques que pratiques, OpenGL ne nous ayant pas vraiment posé de gros problèmes pendant le développement.

Concernant les améliorations que nous aurions aimées mettre en place, mais que nous n'avons pas pu faute de temps, il y a dans un premier temps la correction de la déformation des jouets sur la surface. On sait que cette dernière est due à l'ajout du déplacement sur la trajectoire, et elle est très probablement causée par un mauvais enchaînement des transformations après le dessin des deux jouets. Mais nous n'avons pas réussi à trouver le temps pour corriger ce problème. Il y a également l'animation du bilboquet, qui pourrait largement être améliorée, notre joueur étant actuellement vraiment frileux. On aurait pu mettre en place quelque chose de plus complexe, avec la boule qui retombe en dessous du manche et qui est remontée par le fil, mais comme pour l'amélioration précédente, le temps nous a manqué. Enfin il y a quelques détails qui ne sont pas parfaits. Par exemple, concernant la modélisation de la sphère, nous avons utilisé des quadrilatères pour toutes les faces. Cependant, nous savons qu'en réalité, si on utilise les formules d'Euler sur ces formes, ces dernières ne sont pas constructibles, et paraissent juste correctes sans forcément l'être non plus. Pour une animation en synthèse comme celle-ci, ce détail n'est pas foncièrement gênant, mais c'est une amélioration qui est facilement ajoutable, et que nous aurions mise en place sans problème si nous nous en étions rendu compte plus tôt. A posteriori, nous avons également remarqué que nous avions bien fait de ne pas utiliser les VBO et les shaders, par rapport au temps que le développement nous a déjà pris sans, mais il aurait pu être intéressant, il est vrai, de les mettre en place pour gérer l'éclairage de notre scène.

Nous vous remercions pour votre attention pendant la lecture de ce rapport, en espérant qu'il aura su de manière claire, et tout de même assez précise, présenter le travail effectué lors de ce projet de Synthèse d'Images.