Licence Sciences et Technologies – 3ème année
Module Synthèse d'image

Rapport

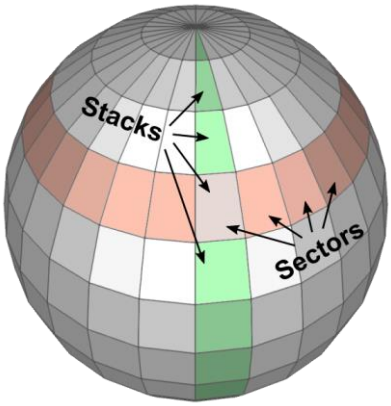
Projet Synthèse d'image

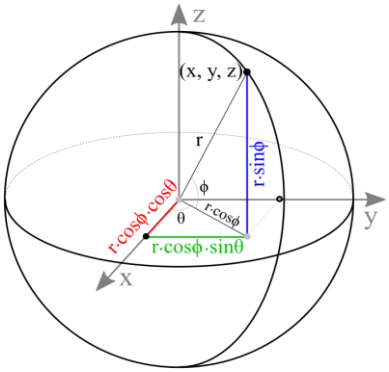
Table des matières

MODELISATION

On utilise 3 formes principales afin de modéliser notre tortue. Les tétraèdres, les sphères et les cylindres. Ces formes sont créées à partir de tableaux de points, ceux-ci nous permettent de créer une matrice regroupant l'indice de la face et l'indice des points correspondants. Puis ces informations sont utilisées pour la modélisation de la forme finale (grâce à la primitive GL_POLYGON pour les sphères et cylindres, et GL_TRIANGLES pour les tétraèdres).
Pour que la modélisation fonctionne, il est important de traiter les points dans l'ordre trigonométrique.

La sphère



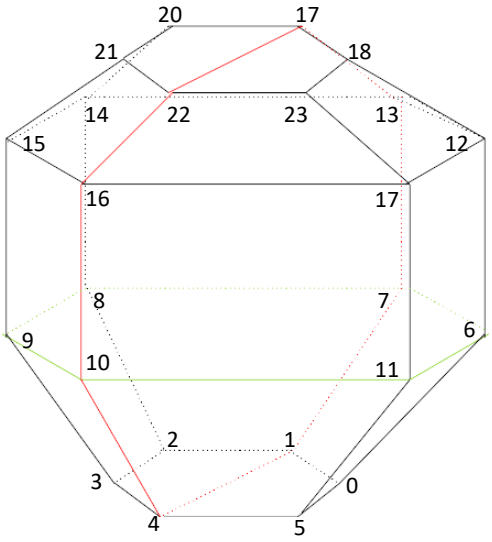


source : http://www.songho.ca/opengl/gl_sphere.html

Afin de modéliser notre sphère, il nous faut connaître plusieurs informations, notamment combien de « méridiens » et de « parallèles ». Ceci permettra de déterminer l'angle qui servira d'intervalle sur lequel on construira chaque point de la sphère (dθ pour les méridiens, dϕ pour les parallèles). Nous nous servirons du système de coordonnée polaire afin de construire les points.

Représentation schématique de la modélisation de la sphère :

Nombre de méridiens : 6
Nombre de parallèles : 4



$$\begin{cases} x(\theta,\varphi) = r * \cos(\theta) * \cos(\varphi) & \theta \in [0,2\pi] \\ y(\theta,\varphi) = r * \sin(\theta) * \cos(\varphi) & \varphi \in [\frac{-\pi}{2},\frac{\pi}{2}] \\ z(\theta,\varphi) = r * \sin(\varphi) \end{cases}$$

$$d\theta = \frac{\text{longueurIntervTheta}}{nombreMéridiens} = \frac{2\pi}{6}$$

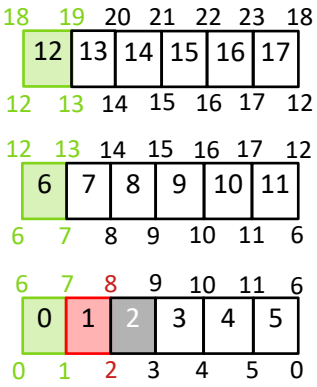
$$d\varphi = \frac{\text{longueurIntervPhi}}{nombreParallèles} = \frac{\pi}{4}$$

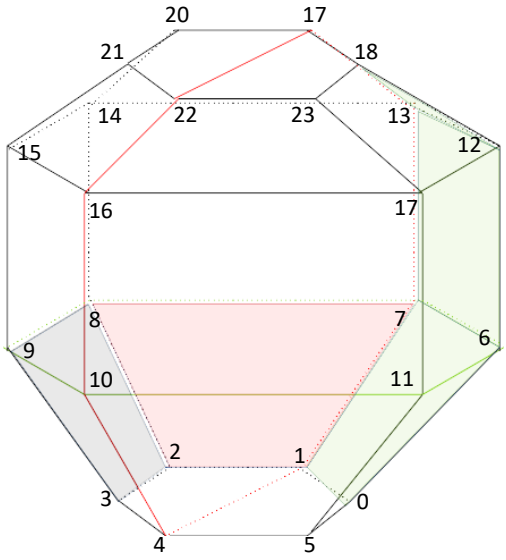
θ = débutIntervTheta + j * dθ
φ = débutIntervPhi + i * dφ (i et j les compteurs de boucle)

Construction des points :

Les points sont donc placés dans un tableau, qui sera utilisé par la suite afin de numéroter les faces. Ces deux étapes serviront ensuite à modéliser notre sphère.

Les faces :





Ici, l'avantage est que le sens de lecture est le sens trigonométrique. Avec cette méthode on peut établir facilement une automatisation de la numérotation des faces.

Le cylindre

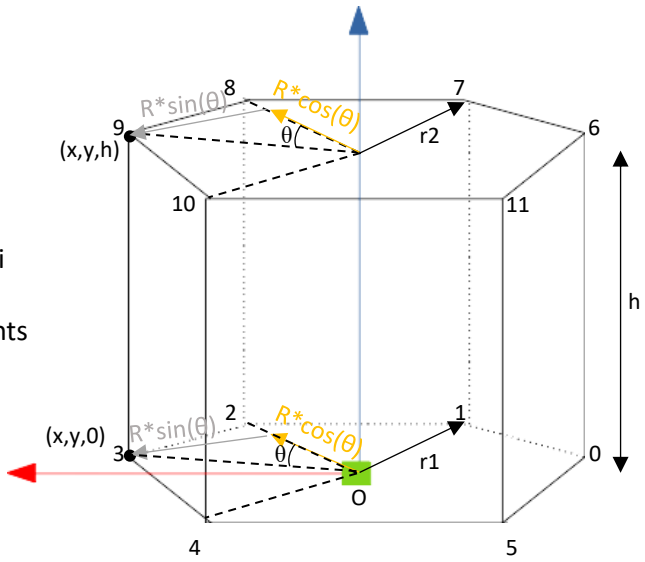
Représentation schématique de la modélisation du cylindre :

Pour n = 6 :

$$\begin{cases} x(r,\theta) = r * \cos(\theta) & \theta \in [0,2\pi] \\ y((r,\theta) = r * \sin(\theta) \\ z = 0 \; ou \; h \end{cases}$$

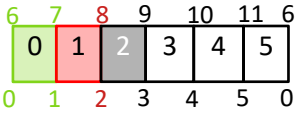
avec θ = (2πi)/n
i est le compteur de boucle, n est le nombre de section

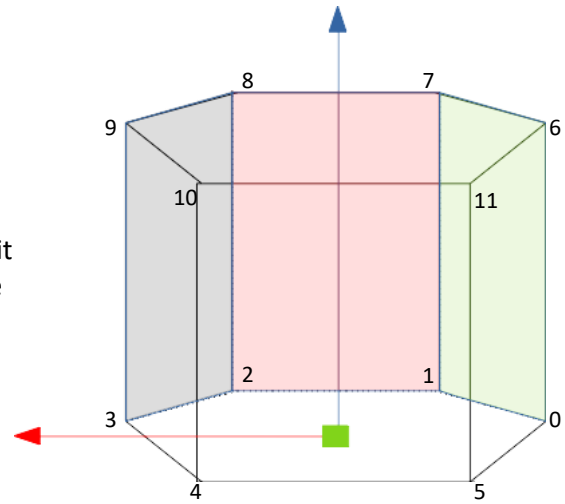
Les indices des points commencent à 0 sur l'axe z, puis sont construits de h plus haut sur cet axe.



Nous avons deux méthodes pouvant être appelées qui servent à la construction d'un cylindre, l'une construit tous les points du cylindre, l'autre s'appuie sur les points d'un cylindre précédent, pour se construire à la suite de ce dernier.

Indices des faces :





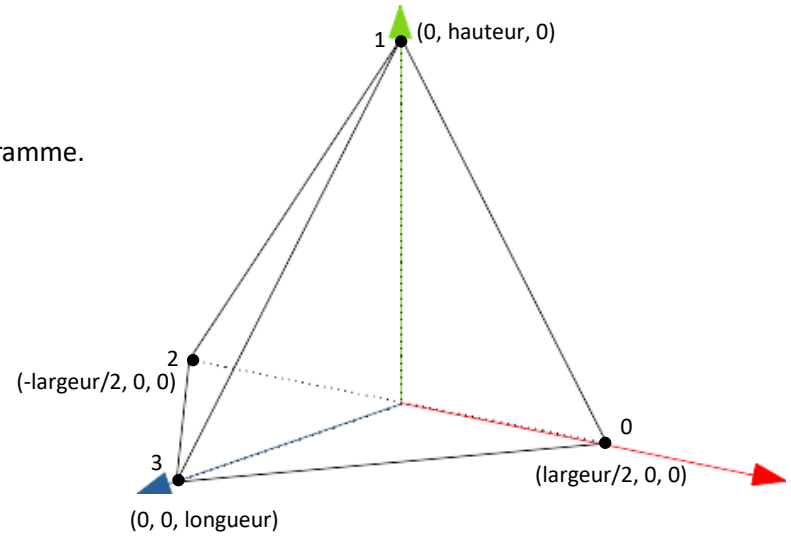
Même principe que pour la sphère, quelque soit l'orientation de la face, le sens trigonométrique sera toujours le même.

Le tétraèdre

Le tétraèdre est la figure la plus simple à modéliser dans ce projet. Elle ne nécessite que 4 points, et ils sont entrés en dur dans le programme, et non calculé grâce à une formule.

Les faces sont aussi entrées en dur dans le programme.

Face 0 : points 0, 1, 2
Face 1 : points 0, 3, 2
Face 2 : points 0, 1, 3
Face 3 : points 3, 1, 2



Cette fois-ci, on utilise la primitive GL_TRIANGLES afin de modéliser ce solide.

Modélisation des membres

Les solides qui serviront de base à la création de notre tortue sont maintenant modélisés. Il faut donc les déplacer entre eux de manière à créer des formes géométriques plus complexes. Nous allons donc nous concentrer sur la modélisation du bras et de la jambe de la tortue.

1.Le bras

Le bras est composé de plusieurs formes géométriques, d'une série de cylindres construit à la suite, d'une sphère (pour l'épaule), ainsi que 3 tétraèdres pour les doigts.

Les cylindres sont construits depuis l'origine, puis ajoutés les uns à la suite des autres. Le premier cylindre est construit grâce à la méthode « draw_cylindre », puis la méthode « draw_cylindre_avec_cylindre » construit les autres en utilisant les n (nombre de points sur le « couvercle » du cylindre) premiers points du tableau de point, puis construit les n points restant, en décalant ces points de h (hauteur du cylindre) sur l'axe z.

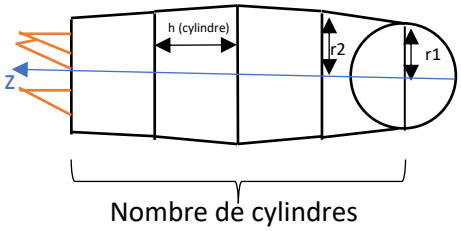
Ceci permet d'utiliser les instances des points des cylindres précédents et ne pas refaire les même calculs plusieurs fois de suite, puis de former un bras plus « arrondi », plutôt qu'un simple cylindre.

Concernant les doigts de la tortue, ce sont des tétraèdres modélisés depuis l'origine, puis déplacé sur l'axe z de h (hauteur de chaque cylindres) *nombreDeCylindres.

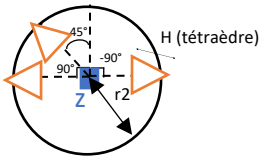
Ensuite, deux des doigts sont pivotés de 45° et 90° sur l'axe z, et déplacés sur leur axe y du rayon r2 du dernier cylindre – la hauteur du doigt. Le dernier doit est pivoté de -90° sur l'axe z, puis subit la même transformation.

<u>De façon schématique :</u>

Bras vue de profil :



Bras vue de face :

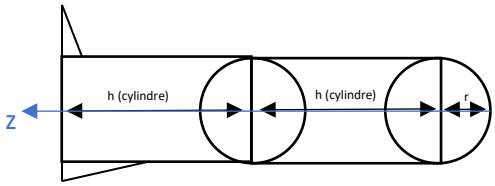


2.La jambre

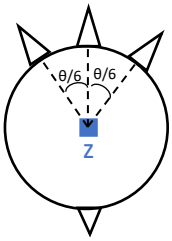
La jambe a été modélisée « à plat » le long de l'axe z afin de faciliter la modélisation. La jambe est composée d'une sphère servant d'articulation à la cuisse, puis d'un cylindre (les deux modélisés à l'origine). Puis l'on créé une autre sphère que l'on déplace d'une distance h (hauteur du cylindre) le long de l'axe z. Puis on modélise le tibia, qui sera un cylindre également, et que l'on déplace de h sur z. Puis on modélise les griffes, pour cela, trois griffes sont déplacées de 2*h le long de z pour arriver au bout de la jambe. Puis l'on dispose les griffes de façon à ce qu'elles soient pivotées et espacées d'un angle $\pi/6$ entre elles. La dernière griffe est située sur le talon.

<u>De façon schématique :</u>

La jambe vue de côté :



La jambe vue de face :



ASSEMBLAGE

Le buste

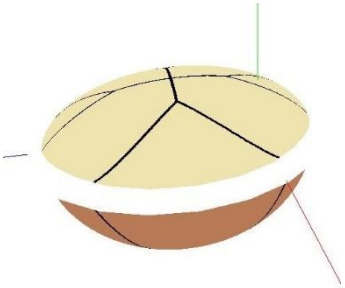


Figure 1 Carapace et ventre redimensionné

Le buste est composé de deux demi-sphères l'une ayant subi une rotation de 90° sur l'axe Z. Donc l'une des deux demi-sphères existe uniquement avec des coordonnées y positive (Le ventre), l'autre n'existe qu'avec des coordonnées négatives (La carapace). Il était important que faire le buste comme deux demi-sphères car cela nous permet de redimensionner le ventre différemment de la carapace. Ce qui donne un visuel un peu plus réaliste car la carapace est plus imposante que le reste du corps. Le ventre a été rendu plus plat et moins large que la carapace. La carapace quant à elle est plus bombée. Un tore a été généré et épouse les contours de la carapace pour dessiner les bordures de cette dernière.

La tête

Figure 2 Buste et tête

La tête est un objet Sphère que nous avons modélisé et texturé. Elle est générée à l'origine, manipulé avec quelques rotations pour l'aligner (la tête fait face à l'axe x sans transformations) et déplacé à l'extrémité la plus éloignée de l'origine du torse.

Les membres

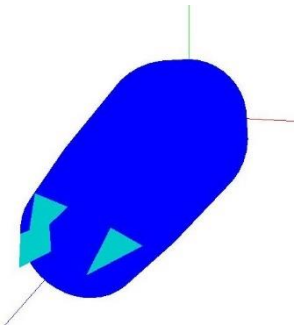


Figure 3 Modélisation bras seul

Les membres (bras et jambes) ont été construit à l'aide de cylindres, de sphères et de tétraèdres. Comme toutes nos structures, les membres sont modélisés avec leur base à l'origine.

Nous n'avons modélisé qu'une jambe et un bras puis nous l'avons cloné pour faire l'autre. S'ensuit une succession de rotation, et translation pour les placers à leur position par rapport à la carapace déjà formée. Il est à noter que comme les membres sont modélisés après la tête texturée, la couleur de la texture de la tête est déjà chargée pour les membres. C'est ainsi que nous avons exactement la même couleur que la tête.

La tortue est maintenant modélisée mais elle est posée sur sa carapace et est encore bien trop imposante. Donc pour finir nous avons fait une rotation sur l'axe x, puis sur l'axe z et une mise à l'échelle général de l'ensemble de la structure.

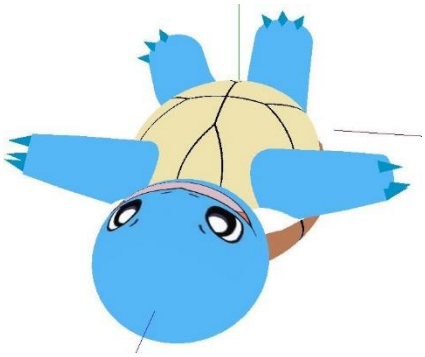


Figure 5 Carapuce sur le dos

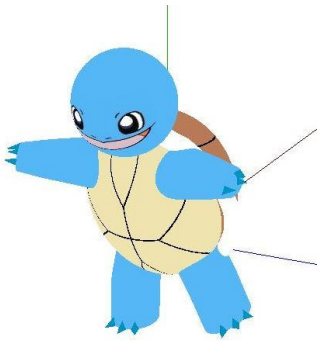


Figure 4 Carapuce final sans lumière

TEXTURE

La gestion

Les textures sont gérées par la Classe Texture. Cette dernière a été créée pour générer et sauvegarder différentes textures pour une utilisation ultérieure. Son constructeur prend comme information la taille de l'image qui sera chargée (largeur et hauteur), ainsi que le chemin d'accès à l'image. Elle va alors lire l'image et stocker les informations dans un tableau de type ***unsigned char*** de taille [***largeur*hauteur***]. L'intérêt de cette façon de faire et que nous n'avons besoin de lire nos textures qu'une seule fois au début du programme, puis nous pouvons charger l'image sauvegardée par chaque texture avec un **Texture::*loadTex()***. Ainsi nous avons chargé toutes nos textures comme variable globale au début de notre fichier principal. Toutes les textures utilisées sont disponibles dans le dossier textures du projet. La texture « null » sert de remise à zéro. Toutes les textures ont été soit dessiné soit adapté par nous.

Les méthodes

getTexCoords

Pour appliquer nos textures sur nos formes composées de faces, nous avons eu besoin d'une fonction pour découper notre texture et distribuer les morceaux de façon cohérente tout de long de la construction. Il s'agit exactement du rôle de **Texture ::*getTexCoords(actualFace,nbL,nbC)*** .

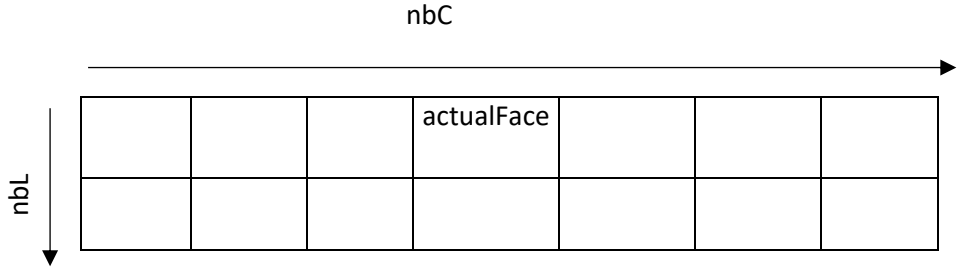


Figure 6 Description schématique getTexCoords

Cette fonction retourne un tableau de float à quatre cellules. {***minimumU, maximumU, minimumV, maximumV***}. Cela permet de savoir où commence et ou finit les coordonnées (entre 0.0 et 1.0) de la portion de texture qui réfère à actualFace. Ainsi la texture est procéduralement appliquée sur chaque face de nos formes.

loadJpegImage

Une fonction appelée dans le constructeur par défaut. Elle va vérifier le type d'image, ses dimensions, son format et si toutes les informations sont conformes, elle va transformer l'image en tableau de type ***unsigned char*** qui sera rangé dans l'attribut ***image*** de l'objet ***Texture***. Cette fonction peut être rappelée manuellement par la suite, mais il faudra que la nouvelle image chargée soit de dimension similaire à la précédente. Pour lire l'image, nous utilisons la librairie jpeg.

loadTex

Une fois l'image transformée, il faut encore la charger en tant que texture. C'est ce que fait **loadTex**. Cette fonction récupère le tableau d'informations, la largeur et la hauteur de l'image et va simplement appeler glTexImage2D et renseigner toutes les informations. La texture peut être utilisée par la suite. Cette fonction est automatiquement utilisé dans la fonction de l'objet **Sphere::*drawSphere(Texture t)***. Pour appliquer les textures lors de la construction des faces.

Sol texturé

Nous avons ajouté un sol avec des coordonnées en dur dans le programme. Le sol est également texturé en dur.

LUMIERE

Nous avons utilisé les lumière LIGHT0, LIGHT1 et LIGHT2. Toute jouent un rôle particulier dans la scène.

LIGHT0

La lumière par défaut de notre scène. Elle tourne autour de notre objet (détails dans la partie animation). Elle illumine de façon AMBIANTE et avec une lueur blanche. Nous avons aussi préparé nos formes à recevoir la lumière avec ***glNormal3f***. (Activable/Désactivable avec la touche '0')

LIGHT1

Une lumière qui ne se déclenche que lors de l'animation (touche '1' du clavier). Elle éclair de façon AMBIANTE avec une lueur bleu. Elle suit la bulle d'eau formé par l'animation et disparait avec elle.

LIGHT2

Une lumière qui illumine en SPOTE sur le bas de notre tortue. Elle est teintée en rouge. Cette lumière n'a pas de cohérence dans la scène. Elle est là pour ajouter un autre type de lumière. (Activable/Désactivable avec la touche '2').

La classe Normal

Afin que les lumières fonctionnent, il faut définir à nos solides des vecteurs normaux sur chacune de leurs faces. Ceci permettra à opengl de déterminer quelles faces sont éclairées, et quelles faces sont ombragées.

On définit ces vecteurs normaux à la création de chacune des formes géométriques que l'on construit. Le constructeur de cette classe prend en paramètre 3 points d'une face donnée (points A, B et C), calcul les vecteurs $\overrightarrow{AB}$ et $\overrightarrow{AC}$, et enfin effectue le produit vectoriel de ces deux vecteurs, ce qui donne le vecteur normal. Ensuite, il faut que ces vecteurs normaux aient une norme comprise entre 0 et 1, donc on divise la norme de ce vecteur normal avec chacune de ses coordonnées.

ANIMATION

Les animations dépendent toutes d'une variable globale qui va déterminer le timing de l'animation et son état.

Le soleil

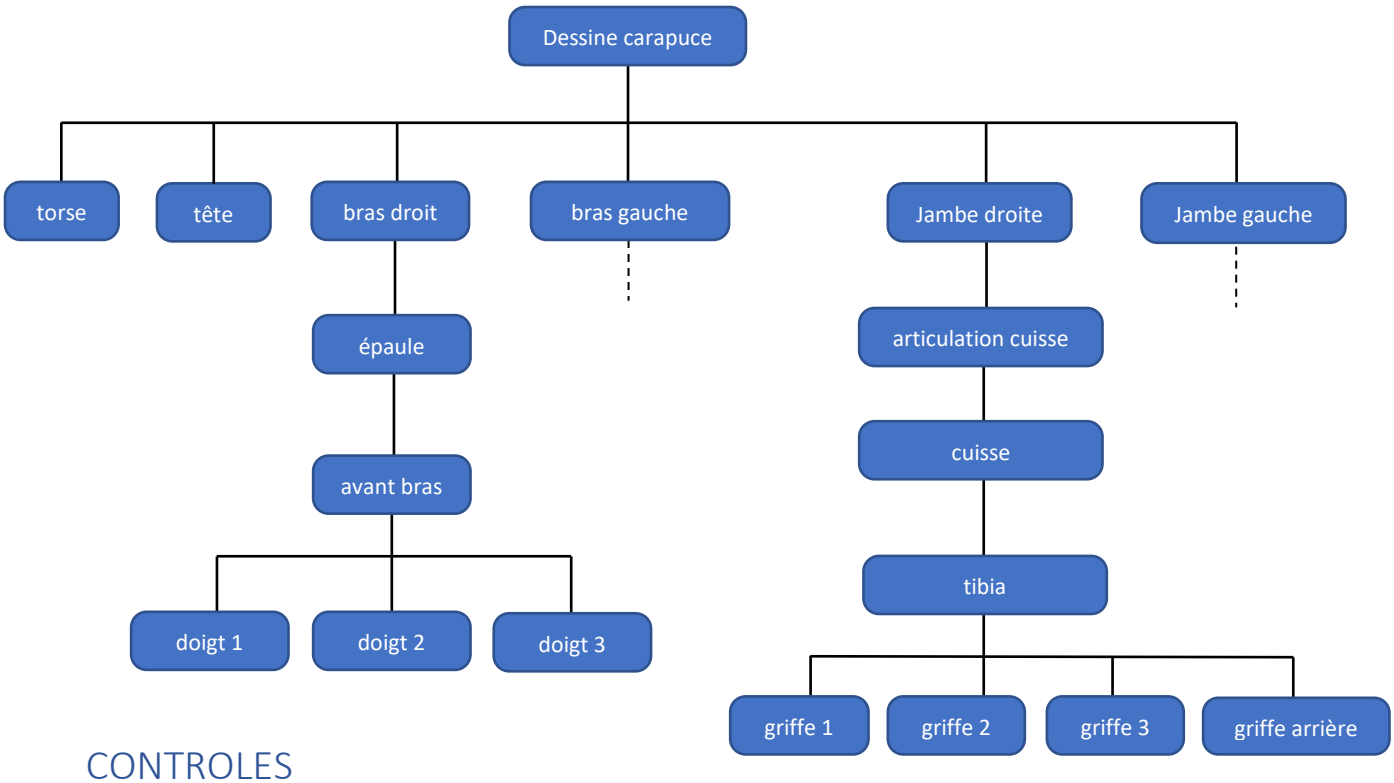
LIGHT0 se déplace automatiquement autour de notre construction. Pour se faire elle saute de point en point sur un cercle généré en amont. (Ces points sont modélisés au niveau de la tête de notre tortue). Les points sont calculés une seule fois et stocké dans la variable globale coordSoleil. A chaque top d'animation, le soleil passe à la coordonné suivante. S'il arrive au bout du tableau il retourne au premier point. Tout ceci est géré par la fonction ***moov***, qui elle-même est passé en paramètre de ***glutIdleFunc***.

La bulle d'eau

Notre seconde animation est une bulle d'eau qui sort de la bouche de notre tortue. En réalité cette bulle existe en permanence dans la tête de notre tortue elle est agrandie et déplacé au moment de l'animation. De nombreuses choses se passent pendant l'animation. La bulle grossit et se déplace, mais le buste, les bras et l'une des jambes suivent le mouvement. Nous avons rencontré quelques difficultés sur cette partie car notre hiérarchie n'était pas très bien optimisée. Nous avons donc dû déplacer la jambe, et les bras manuellement.

Toute l'animation se fait sur 40 tops. Les 20 premiers influencent l'animation montante (Le recule), les 20 derniers influencent l'animation de retour. La bulle d'eau est animée en animation montante sur les 40 tops. Elle retourne dans la bouche de la tortue à la fin du 40ème. Une seule bulle d'eau peut exister en même temps. Lorsqu'elle se termine, elle remet toutes les informations à jours pour la prochaine animation. Il est a noté que cela n'arrête pas l'animation du soleil.

Graphe hiérarchique des articulations de la tortue



CONTROLES

Déplacements dans la scène 3D

N'ayant pas réussi à détecter les flèches directionnelles du clavier, nous avons décidé de placer les mouvements de caméra sur « f »(gauche) « t »(haut) « g »(bas) et « h »(droite). Afin de pivoter la scène grâce à ces contrôles, nous incrémentons et décrémentons les variable anglex ou angley, ce qui pivotera l'ensemble des constructions dans la scène. En effet, les variables anglex et angley sont utilisées au début de la fonction « affichage() » dans une fonction « glRotatef() », ce qui implique qu'à chaque changements de valeur de anglex ou angley, les constructions de la scène pivoteront.

Zoom

Le zoom a été implémenté sur les touches 'z'(zoom avant) et 'Z'(zoom arrière). Ici on utilise la fonction glOrtho, qui effectue une projection orthographique dans un plan 2D qui représente l'écran d'ordinateur ou la fenêtre dans laquelle est affichée la scène. Les méthodes « zoomIn() » et « zoomOut() » font évoluer les variables « base_ortho_x » et « base_ortho_y » en fonction du « zoomFactor »(fixé à 0.1). Après évolution des variables, on les remet en paramètre de la fonction « glOrtho() », ce qui recalculera la projection orthographique et redessinera la scène 3D en 2D. Il est important que les deux premiers arguments de « glOrtho » soient des valeurs inverses (même valeur numérique mais de signe opposé), afin que l'image ne soit pas déformée.