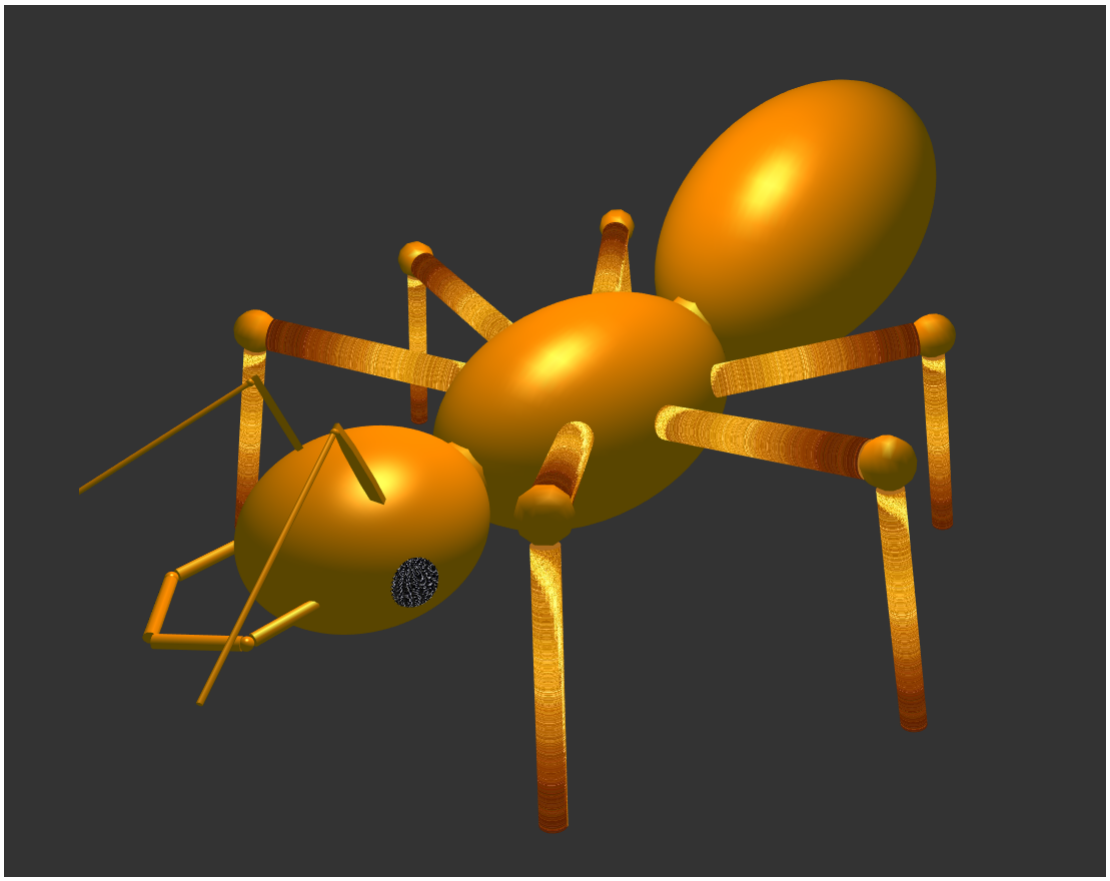
Rapport Synthèse d'Image

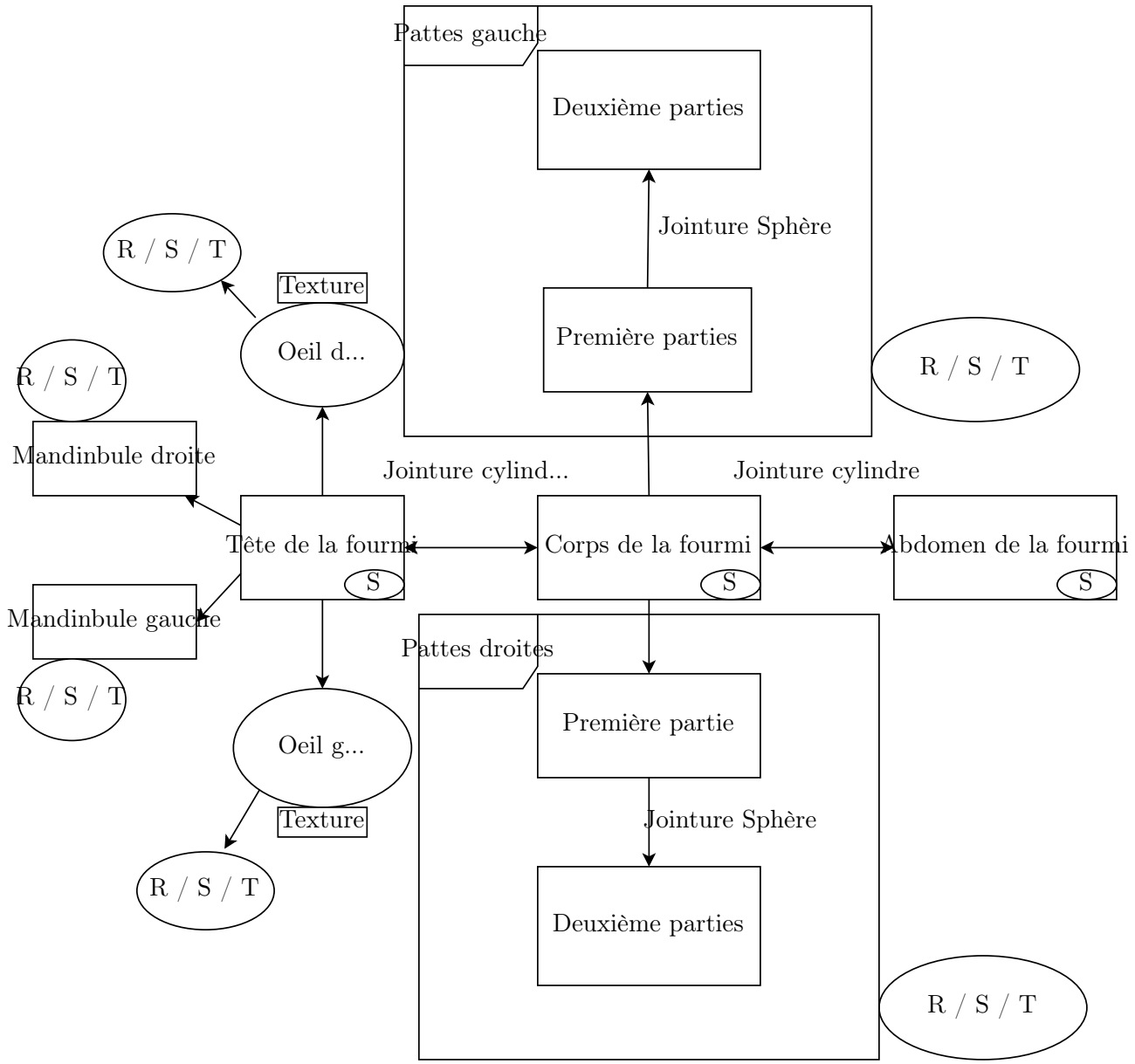
Yann TROU et Wassim DJELLAT

Novembre 2020

Table des matières



Voici notre fourmi, nous avons tenté de créer une fourmi non pas réaliste mais une fourmi originale. Nous avons mis des textures dorées pour les pattes et texture d'oeil de fourmi pour les yeux.



R signifie Rotate / S signie Scale / T signifie Translate.
C'est un schémas descriptif de la conception de notre fourmi.

2. Primitives

Pour les primitives, nous avons choisi le cylindre et la sphère. Les cylindres ont été utilisé pour créer les 6 pattes de la fourmi et biensur nous avons entouré ces pates de la texture doré.
Pour les sphères, nous les avons utilisés pour former les yeux. Sur ces yeux nous avons plaqué la texture d'oeil de fourmi.

III | Eclairage

Pour ce projet, il nous est demandé de gérer deux types de lumières. Nous avons donc choisi de gérer un spot et une lumière ambiante que nous allons détailler ci-dessous.

1. Lumière ambiante

La 1ere lumière que nous avons choisi de gérer est une lumière ambiente. Elle se situera sous la fourmi et sera de couleur verte (comme si la fourmi était sur du gazon).
Ses paramètres sont les suivants :

- position : x=0, y=-5, z=0
- couleur : r=0, g=0.4, b=1.0, a=1.0

Ci dessous, le code représentant la lumière ambiente.

```
1        /*Lumiere 1: ambiente verte en dessous de la fourmi*/
2        posAmbient[0] = 0;
3        posAmbient[1] = -5;
4        posAmbient[2] = 0;
5        posAmbient[3] = 0;
6        GLfloat ambient2[] = { 0.0, 0.4, 0.0, 1.0 };
7        glLightfv(GL_LIGHT1, GL_AMBIENT, ambient2);
8        glLightfv(GL_LIGHT1, GL_POSITION, posAmbient);
9        glEnable(GL_LIGHT1);
```

2. Spot

La 2e lumière que nous utilisons est un spot de couleur blanche située au dessus de la fourmi, comme la lumière du soleil.
Ses paramètres sont les suivants :

- position : x=0, y=5, z=0
- couleur (ambiente) : r=0.3, g=0.3, b=0.3, a=0.3
- couleur (diffuse) : r=1, g=1, b=1, a=1
- couleur (spéculaire) : r=1, g=1, b=1, a=1
- angle d'éclairage : 90 degrés
- atténuation linéaire : 0.1
- atténuation quadratique : 0.01
- atténuation constante : 1.0

Ci dessous, le code représentant le spot.

```
1        /*Lumiere 0: spot blanc au dessus de la fourmi*/
2        posProj[0] = 0;
3        posProj[1] = 5;
4        posProj[2] = 0;
5        posProj[3] = 1.0; //important pour definir un spot
6        GLfloat dirProj[] = { 0.0, 0.0, 0.0, 1.0};
7        GLfloat ambient[] = { 0.3, 0.3, 0.3, 1.0 };
8        GLfloat diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
9        GLfloat specular[] = { 1.0, 1.0, 1.0, 1.0 };
10       glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
```

```
11          glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
12          glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
13          glLightfv(GL_LIGHT0, GL_POSITION, posProj);
14          glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dirProj);
15          glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 90.0);
16          glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.1);
17          glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.01);
18          glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 1.0);
19          glEnable(GL_LIGHT0);
```

IV | Textures

D'après le cahier des charges, il nous faut deux textures, une plaquée et une entourée. Nous avons donc deux textures s'appliquant sur les primitives. Aussi, comme nous utilisons des primitives de freeglut, nous avons simulé les réflexions de l'or face à la lumière.

1. Textures sur les primitives

Déjà comme nous avons fait le projet sur visual studio, il nous est impossible d'utiliser libjpeg pour charger des images et les utiliser. La solution que nous avons trouvé est la bibliothèque stb_image que Yann avait vu lors de projets personnels avec openGL (c'est nous qui avons donné cette solution aux autres d'ailleurs). Cette bibliothèque tient dans un seul (gros) fichier d'en tête, et permet de charger n'importe quelle image en une simple ligne de code. La méthode loadJpegImage() du cours a ainsi été transformée en la méthode :

```
1       unsigned char* loadJpegImage(const char* fichier, int* width, int* height, int*
            bpp)
2       {
3           /*chargement de l'image*/
4           unsigned char* image = stbi_load(fichier, width, height, bpp, 3);
5
6           /*vrification du chargement*/
7           if (image == nullptr)
8               std::cout << "Erreur, impossible de charger l'image " << fichier <<
                    std::endl;
9           else
10              std::cout << "Texture chargee: " << fichier << std::endl;
11
12          return image;
13      }
```

Maintenant, parlons des textures en elles-mêmes. Comme nous avons plusieurs textures, l'utilisation de glBindTexture devient obligatoire.
Le code suivant représente le chargement et paramétrage de la texture pour la sphère (identique à celui pour charger la texture des cylindres) :

```
1       glGenTextures(1, &id_tex_sphere);
2       glBindTexture(GL_TEXTURE_2D, id_tex_sphere);
3       tex_sphere = loadJpegImage("noir.png", &width_tex_sphere, &height_tex_sphere,
            &bpp_tex_sphere);
4       glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
5       glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
6       glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width_tex_sphere, height_tex_sphere, 0,
            GL_RGB, GL_UNSIGNED_BYTE, tex_sphere);
7       glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
```

Nous avons deux textures en tout. Déjà, la texture or qui est enroulée autour des cylindres des pattes suivant la formule vue en cours. Ensuite, une autre texture représentant les yeux de la fourmi, plaquée sur les faces de la sphère. Comme cette texture est en trop haute résolution pour la géométrie

et comme nous n'utilisons pas de mipmaps, un effet de scintillement apparait que nous avons décidé de conserver.

2. Matériaux

Comme nous utilisons des solides de glut, leur donner un aspect ressemblant les textures nous a semblé une évidence. Nous avons donc crée un matériaux ayant les réflexions de l'or à partir des valeurs vues dans le cours. Le matériau a les propriétés suivantes :

- couleur(ambiante) : r=0.24725, g=0.1995, b=0.0745, a=1.0
- couleur (diffuse) : r=0.75164, g=0.60648, b=0.22648, a=1.0
- couleur (spéculaire) : r=0.628281, g=0.555802, b=0.366065, a=1.0
- réflexion : 51.2

```
1        /*application des rflections de matriau (type OR)*/
2        glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
3        GLfloat ambientOr[] = { 0.24725, 0.1995, 0.0745, 1.0f };
4        GLfloat diffuseOr[] = { 0.75164, 0.60648, 0.22648, 1.0f };
5        GLfloat specularOr[] = { 0.628281, 0.555802, 0.366065, 1.0f };
6        GLfloat shine = 51.2;
7        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambientOr);
8        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, diffuseOr);
9        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specularOr);
10       glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, shine);
```

les animations

1. Les animations automatiques

Les animations automatiques concernent les éléments suivants :
— Tête : mouvement de gauche à droite (de -15 degrés à 15 degrés)
— Mandibules : mouvement de l'intérieur vers l'extérieur (de -10 à 0 degrés
— Queue : mouvement de bas en haut (de -10 à 10 degrés)
— Antennes : mouvement de droite à gauche rapide. (de -10 à 10 degrés)

Pour faire une animation, nous avons deux variables : la 1ère, définit le sens de l'animation. La 2e est la valeur de l'animation, généralement l'angle de la rotation.
Le sens de l'animation est régulé avec des seuils qui sont définis dans le programme.

2. Les animations manuelles

Pour l'animation manuelle, on a définit les fonctions suivantes pour déplacer les pattes de la fourmi :

$$10 * cos(x * pi/2)$$

—

$$10 * cos(x * pi/3)$$

—

$$10 * cos(x * 2pi/3)$$

—

$$10 * sin(x * pi/2)$$

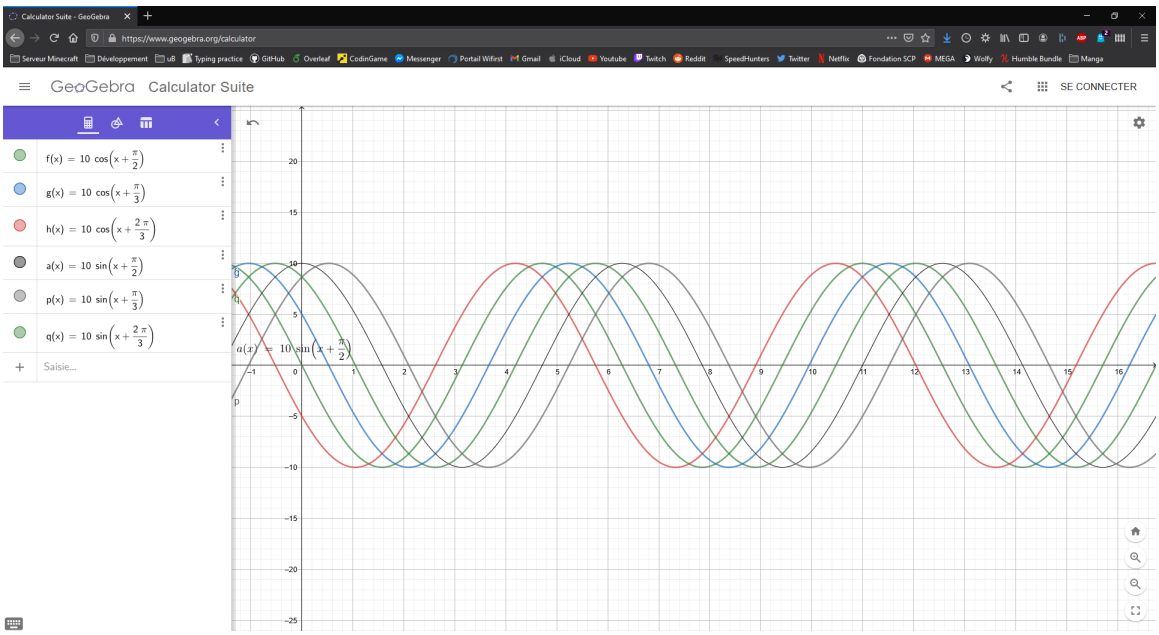
—

$$10 * sin(x * pi/3)$$

—

$$10 * sin(x * 2pi/3)$$

Ci desssous, les formules représentées dans geogebra.



L'idée derrière l'utilisation des formules est d'avoir un mouvement fluide mais aussi que la fourmi "galope" en ayant qu'une seule patte au sol à la fois. L'animation manuelle est déclenchée via l'appui de la touche e, détaillé dans la rubrique contrôles.

Les Contrôles

1. Le Zoom

Ce code est pour récupérer les valeurs du Zoom appliqué :

```
        case 'z':  /*Zoom avant*/
        zoom -= 10.0f;
        std::cout << "zoom: " << zoom << std::endl;
        glutPostRedisplay();
        break;

        case 'Z':  /*Zomm arriere*/
        zoom += 10.0f;
        std::cout << "zoom: " << zoom << std::endl;
        glutPostRedisplay();
        break;
```

2. Modification de la vue de l'objet

Ici nous avons la fonction qui permet de permet de modifier la vue de l'objet avec les touches :

```
        void specialKeyInput(int key, int x, int y)
        {
            switch (key)
            {
            case GLUT_KEY_UP:
                angley += 10.0f;
                glutPostRedisplay();
                break;

            case GLUT_KEY_DOWN:
                angley -= 10.0f;
                glutPostRedisplay();
                break;

            case GLUT_KEY_LEFT:
                anglex += 10.0f;
                glutPostRedisplay();
                break;

            case GLUT_KEY_RIGHT:
                anglex -= 10.0f;
                glutPostRedisplay();
                break;
            }
        }
```

3. Touches pour animation

Nous avons créer dans la création des pattes de la fourmi l'animation des pattes avec des formules sin et cos pour faire paraître une fourmi réel.

```
case 'e':
 updateAnimPattes();
 glutPostRedisplay();
 break;
```

4. Les affichages

Pour les affichages nous avons une fonction affichage() qui s'en occupe.
Ce code est pour le rendu du zoom :

```
/*dfinition de la perspective et application du zoom*/
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(zoom, 500.0 / 500.0, 0.1, 100.0);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

Ce code est pour le position de la caméra :

```
/*positionnement de la camra*/
 gluLookAt(0.0, 0.0, 10.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
 glRotatef(angley, 1.0, 0.0, 0.0);
 glRotatef(anglex, 0.0, 1.0, 0.0);
```

ce code est pour les formules de mouvements des pattes gauches :

```
glRotatef(10 * cos(animValue_patte + (M_PI) / 2), 1, 1, 0);
glRotatef(10 * cos(animValue_patte + (M_PI) / 3), 1, 1, 0);
glRotatef(10 * cos(animValue_patte + (2 * M_PI) / 3), 1, 1, 0);
```

ce code est pour les formumles de mouvements des pattes droites :

```
glRotatef(10 * sin(animValue_patte + (M_PI) / 2), 1, 1, 0);
glRotatef(10 * sin(animValue_patte + (M_PI) / 3), 1, 1, 0);
glRotatef(10 * sin(animValue_patte + (2 * M_PI) / 3), 1, 1, 0);
```