| | | |
|---|---|---|
| | **IMERIR**<br>**2Année** | |

# TP : Infographie

## Réalisation d'un mini système solaire

| | |
|---|---|
| A.Rharmaoui<br>Version3 | 20116-17 |

# Réalisation d'une animation 3D

# avec Glut-OpenGL

## 1. Objectif:

Ce projet a pour objectifs la modélisation d'un système «dynamique» en utilisant des transformateurs géométriques. Il s'agit de réaliser une animation simulant un mini système solaire.

## 2 Déroulement:

Ce projet est planifié sur 3 séances comme suit:

**S1:** Travail préparatoire: (suite du TD)
- Rappels: Affichage de primitives avec Glut.
- Gestion de la profondeur (Exemple du TD).
- Éclairage d'une scène 3D.
- Composition de transformations géométriques.

**S2 :** Réalisation de l' animation 3D:
- Mini système solaire.

**S3 :** Finalisation et contrôle des résultats.

## 3. Evaluation:

L'évaluation sera réalisée en mode contrôle continu de tous les exercices.

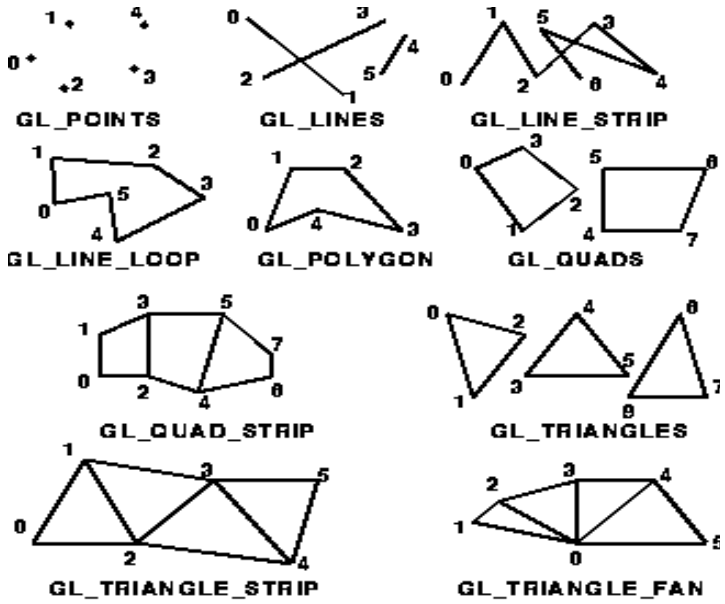
## 4. Documents:

Touts les documents sont disponibles sous le e-learning de l'école.

## **Seance1**:

# Rappel : Dessin de primitives

Pour commencer ce TP un bref rappel des primitives sous OpenGL:




Le début du dessin d'une primitives (vue en TP 1A) se fait avec glBegin (Glenum *mode*) : *mode* indique l'un des types de primitive schématisés ci-dessus. Elle est suivie de la définition des différents points composant l'objet. Chaque point est défini grâce à la fonction glVertex, qui existe sous plusieurs formes : glVertex2d, glVertex2f, glVertex2i, glVertex2s, glVertex3d, glVertex3f, glVertex3i, glVertex3s, glVertex4d, glVertex4f, glVertex4i, glVertex4s, glVertex2dv, glVertex2fv, glVertex2iv, glVertex2sv, glVertex3dv, glVertex3fv, glVertex3iv…
glColor() permet de fixer la couleur courante. De même, elle existe sous de nombreuses formes.
glEnd(), indique la fin de l'entrée de coordonnées.

Exemple d'un carré : ProjetPolygone (polygones.c)

```
….....................

glColor3f(1.0, 1.0, 1.0);

glBegin(GL_POLYGON); //commence le dessin d'un carré
```

```
glVertex3f(-1, -1, 0); //coordonnées du premier sommet
glVertex3f(1, -1, 0);
glVertex3f(1, 1, 0);
glVertex3f(-1, 1, 0);

glEnd();  //fin du tracé du carré
```

## Tests des résultats calculés en TD:

Charger le projet ProjetPlygone C++ sous le dossier Codes Exemples qui permet d'afficher un polygone de type carré: polygones.c

Mettez pour commencer le fond de fenêtre en noir: (0,0,0,1) :

Pour cela modifier les valeurs de la fonction ***glClearColor*** dans la fonction d'initialisation ***GLvoid initGL();***

le modèle de couleur utilisé est le RGBA avec la transparence Alpha A; pour A =1 le rendu est opaque:

Pour afficher les Polygones P1 et P2, résultats de la visualisation 3D calculée en TD. Il faut modifier le code des fonctions suivantes :

***mon_window_display():*** dessine et affiche la scène selon le type de projection spécifiée.

Modéliser les polygones P1 en rouge et P2 en vert:

P1= {S1(50.0,0.0,0.0),S2(50.0,0.0,50.0),S3((150.0,50.0,0.0)}

P2= {S1(100.0,50.0,0.0),S2(100.0,0.0,0.0),S3(100.0f,0.0,50.0)}

en utilisant la primitive GL_POLYGON (exemple ci dessus):

A la fin du dessin (après le ***glEnd()***) OpenGL stocke l'image dans un Buffer et ne l'affiche qu'à l'appel de la commande ***glFlush().***

***mon_window_reshape():*** Cette fonction prend en charge deux fonctionnalités OpenGL de base : le redimensionnement et la projection

Redimensionnement:

Si la taille de la fenêtre est modifiée la fonction ***glViewport(0, 0, width, height)*** permet de prendre en compte les nouvelles dimensions **(*width, height*)**; Ce ***viewport*** est initialisé par les valeurs déclarées dans le main:

***glutInitWindowSize(WIDTH, HEIGHT);***

Projection:

Pour spécifier une projection en 3D, on dispose de deux fonctions:

***glOrtho();***
***gluPerspective()***
Les paramètres de chaque fonction définissent le "volume de vision fini" associé.
Remplacer la ***gluOrtho2D*** par la projection parallèle :
***glOrtho( -200, 200, -200, 200, -100, 200);***

Compiler puis exécuter le programme!!?
S'agit il d'un bug ou tout simplement d'une mauvaise observation?

Pour faciliter l'interprétation, Afficher le repère de la scène:
l'axe u(1,0,0), : Rouge, l'axe v(0,1,0) :vert, l'axe n(0,0,1) : bleu

Identifier la position par défaut de la caméra pour expliquer le rendu obtenu.

## Intégration d'une caméra dans la scéne:

La fonction ***gluLookAt(eyex,eyey,eyez,centerx,centery,centerz,upx,upy,upz)*** permet de placer la caméra (PRP) dans la scène.

Insérer la commande : ***gluLookAt(200,0,0,150,0,0,0,1,0);***
dans la fonction ***mon_window_display()*** avant de commencer le dessin.
La caméra est placée et orientée ainsi suivant l'axe des x (comme spécifiée en TD) :

## Gestion de la profondeur:Depth-Buffer

Le depth-buffer(z_Buffer) permet de résoudre le problème des faces cachées. Il a les mêmes dimensions que le color-buffer. Il s'active grâce à ***glEnable(GL_DEPTH_TEST)*** et se désactive par ***glDisable(GL_DEPTH_TEST)***. Il ne faut pas, bien sûr, oublier de l'initialiser au début du programme avec la commande :

***glutInitDisplayMode(… | GLUT_DEPTH | …).*** Il doit être effacé avec ***glClear(GL_DEPTH_BUFFER_BIT)*** à chaque fois que la scène est redessinée ( dans ***mon_window_display()***).

Activer le test à l'initialisation dans ***: initGL(),***

Compiler puis exécuter le programme.

## Amélioration du rendu:

Affecter au polygone P2 la couleur blanche et à chaque sommet de P1 une couleur comme suit:

S1 : rouge,S2: vert, S3: bleu.

Initialiser le mode de rendu au **GL_SMOOTH** (mode lissé):

***glShadeModel(GL_SMOOTH)*** (dans l'initGL)

Le dégradé de couleur est obtenu par interpolation bilinéaire des couleurs.

On peut bien sûr désactiver cette fonctionnalité avec ***glShadeModel(GL_FLAT),***

le mode FLAT indique à l'OpenGL de remplir le polygone avec la couleur du dernier sommet spécifié.

Ajouter à la scène une sphère de rayon 25 positionnée en (0,0,0) avec une couleur de votre choix par exemple:

```
glColor3f(0.5f,0.0f, 0.5f);
glutSolidSphere(25,50,50);
```

Passer ensuite en projection perspective pour obtenir un rendu plus réaliste de la scène. Cette projection est modélisée sous OpenGL par:

***gluPerspective(Fovy:(angle ), aspect,ZNear,ZFar);***

Spécifier et tester les paramètres.

## Intégration de lumière et d'éclairage.

### 1. Activation

On active l'éclairage OpenGL, avec les commandes :

```
glEnable(GL_LIGHTING) ;          // Active la gestion des lumières
glEnable(GL_LIGHT0) ;            // allume la lampe 0
```

Huit lampes sont disponibles *GL_LIGHTi avec i:0->7*
Pour éteindre, on utilise **glDisable**.
Le nombre maximal de lumières est obtenu en appelant :
***glGetIntegerv(GL_MAX_LIGHTS, &nb_li***ghts)

### 2. Paramètres

Pour plus de réalisme, les sources modélisables avec OpenGL disposent de plusieurs paramètres qui n'ont pas forcément d'équivalent réel (voir le paragraphe suivant extrait du man d'OpenGl). Une chose par contre ne change pas : la couleur visible d'un objet

dépend à la fois de sa couleur propre et celle de la lampe, et l'intensité de la lumière qu'il reçoit dépend de l'angle de la surface avec la direction vers la source lumineuse comme on l'a vu en cours.

### 3. Fonctions OpenGL

Les fonctions suivantes permettent de spécifier tous les paramètres (Chapitre glLight du man):

- Pour une source lumineuse :

***void glLightf( Glenum light, Glenum pname, GLfloat param )***
***void glLighti( Glenum light, Glenum pname, GLint param )***

light = GL_LIGHTi
pname = GL_SPOT_EXPONENT,
GL_SPOT_CUTOFF,
GL_CONSTANT_ATTENUATION,
GL_LINEAR_ATTENUATION, ou
GL_QUADRATIC_ATTENUATION
et param = la nouvelle valeur

***void glLightfv( GLenum light, GLenum pname, const Glfloat *params)***
***void glLightiv( GLenum light, GLenum pname, const GLint *params)***

light = GL_LIGHTi
pname = GL_AMBIENT, GL_DIFFUSE,
GL_SPECULAR, GL_POSITION,
GL_SPOT_DIRECTION, GL_SPOT_EXPONENT,
GL_SPOT_CUTOFF,
L_CONSTANT_ATTENUATION,
GL_LINEAR_ATTENUATION, ou
GL_QUADRATIC_ATTENUATION
param = la nouvelle valeur

Attention à la position de la lampe : comme toutes les coordonnées entrées, elles sont immédiatement multipliées par la matrice ModelView. Si celle-ci change, il faut repositionner les lampes.

- Pour un matériau :

Les propriétés de surface d'un polygone peuvent être différentes suivant la face que la face est « front » ou « back » (voir glFrontFace, ou le paragraphe sur le culling :

élimination des faces cachées).

***void glMaterialfv( GLenum face, GLenum pname, const GLfloat *params )***
***void glMaterialiv( GLenum face, GLenum pname, const GLint *params )***

| | |
|---|---|
| Face | Specifies which face or faces are being updated. Must be one of GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK. |
| Pname | Specifies the material parameter of the face or faces that is being updated. Must be one of GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION, GL_SHININESS, GL_AMBIENT_AND_DIFFUSE, or GL_COLOR_INDEXES. |
| Params | Specifies a pointer to the value or values that pname will be set to. |

Ajouter à la scène l'éclairage défini par :

```
 void Init_light()
{
                    //différents paramètres
        GLfloat ambient[] = {0.15f,0.15f,0.15f,1.0f};
        GLfloat diffuse[] = {0.5f,0.5f,0.5f,1.0f};
        GLfloat light0_position [] = {0.0f, -10.0f, 0.0f, 0.0f};
        GLfloat specular_reflexion[] = {0.8f,0.8f,0.8f,1.0f};
        GLubyte shiny_obj = 128;

                //positionnement de la lumière avec les différents paramètres
        glEnable(GL_LIGHTING);
        glLightfv(GL_LIGHT0,GL_AMBIENT,ambient);
        glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuse);
        glLightfv(GL_LIGHT0,GL_POSITION,light0_position);
        glEnable(GL_LIGHT0);

                //spécification de la réflexion sur les matériaux
        glEnable(GL_COLOR_MATERIAL);
        glColorMaterial(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE);
        glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT,ambient);
        glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,diffuse);
        glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,specular_reflexion);
        glMateriali(GL_FRONT_AND_BACK,GL_SHININESS,shiny_obj);
}
```

N'oublier pas de déclarer la fonction. L'appel se fait ensuite à l'initialisation dans le main.

Notez qu'ici, on a utilisé 2 fois les tableaux "ambient" et "diffuse". Ceci n'est pas du tout obligatoire. On le fait pour simplifier le problème. Il y a une multitude d'effets différents, donc, testez vous_même. Notez également qu'en ce qui concerne la réflexion sur les matériaux, vous pouvez changer ces propriétés avant de dessiner chaque objet, qui aura

alors des caractéristiques lumineuses propres. (le bois ne reflète pas la lumière comme le métal...)

### Transformations :Rotation et translation

Pour bouger un objet on dispose de deux fonctions prédéfinies (Vues en 1A) :

***void glRotated( GLdouble angle, GLdouble x, GLdouble y, GLdouble z )***

***void glRotatef( GLfloat angle, GLfloat x, GLfloat y, GLfloat z )***

***void glTranslated( GLdouble x, GLdouble y, GLdouble z )***

***void glTranslatef( GLfloat x, GLfloat y, GLfloat z )***

Ces fonctions multiplient à droite la matrice de vue(GL_MODELVIEW). Il faut donc faire attention à l'ordre dans lequel les transformations sont entrées : c'est la dernière transformation entrée qui sera appliquée en premier.

Appliquer à la sphère la translation de vecteur(100,0,0);

Applique au polygone P2 la rotation de 45° autours de l'axe des y

Vérifier la non commutativité de ces 2 transformations.

***Comment animer les primitives indépendamment l'une de l'autre ?***

## Pile de matrice

On va utiliser comme en 1A les piles auxquelles sont associées les matrices. Les fonctions permettant cette manipulation sont :

***void glPushMatrix( void )***

***void glPopMatrix( void )***

qui respectivement empile et dépile la matrice courante. Même s'il n'y a que deux fonctions, il existe bien trois piles différentes (une par matrice). Comme pour les transformations, seule la matrice active peut-être empilée / dépilée (ce qui est logique, puisque que une matrice est empilée pour la sauvegarder avant une transformation).

Le plus souvent, un objet est caractérisé par ses angles de rotation et sa position (x, y, z). Son affichage est alors du type :

```
glPushMatrix() ;
```

```
        glTranslate3f(x,y,z) ;
        glRotate3f(angle,axex,axey,axez) ;
        glBegin(…)
              …
        glEnd() ;
    glPopMatrix() ;
```

Modifier le programme pour rendre la sphère indépendante de la rotation du polygone P2.

Réaliser l'animation simple définie par:

la rotation de la sphère d'un angle teta autour de l'axe des y suivie de la translation de vecteur(100,0,0).

Déclarer teta en variable globale (??? cette solution est la plus simple),

associer ensuite à la touche 'r' l'incrémentation de teta comme suit:

```
…....
case 'r':
        teta = teta+1; //penser à limiter teta à 2pi
        glutPostRedisplay();// affichage immédiat
break;
…..........
```

Pour améliorer le rendu on affiche l'animation en double Buffer. Initialiser pour cela :

***glutInitDisplayMode(GLUT_DOUBLE |GLUT_RGBA |GLUT_DEPTH);***

et on ajoute à la fin de la fonction d'affichage la commande :***glutSwapBuffers().***

Écrire la fonction permettant le déplacement de la caméra sur une sphère de rayon r=200 (cordonnées sphériques, rayon :r, angle1/z, angle2/y) en utilisant les touches:

Up - Flèche Haut : incrémente angle1 de 1°;
Down - Flèche Bas : décrémente angle1 de 1°;
Left - Flèche Gauche : incrémente angle2 de 1°;
Right - Flèche Droite: décrémente angle2 de 1°

# S2
# Réalisation d'un mini système solaire
-

Le but de ce TP est de mettre en pratiques les notions acquises pour réaliser un mini système solaire simplifié.

Voici sa structure:

- Dessin du soleil
- PushMatrix
    - Translation dans le repère local d'une planète
    - Dessin de la planète
    - PushMatrix
        - Translation dans le repère local d'une lune
        - Dessin de la lune
    - PopMatrix
    - PushMatrix
        - Translation dans le repère local d'une  autre lune
        - Dessin de la lune
    - PopMatrix
- PopMatrix
- PushMatrix
    - Translation dans le repère local d'une autre planète
    - Dessin de la planète
    - PushMatrix
        - Translation dans le repère local d'une lune
        - Dessin de la lune
    - PopMatrix
- PopMatrix
- etc.

ATTENTION : la moindre erreur, le moindre oubli d'un glPopMatrix() ou glPushMatrix() génère un bug difficile à détecter si on déplace la caméra.

Pour simplifier la réalisation on ne concidère qu'une représentation partielle du système solaire :

- le soleil
- mercure
- vénus
- la terre
- et la lune.

Dimensions des planètes ("proportionnelles" aux tailles réelles):

Soleil : 109
Mercure :8.8
Vénus : 14.4
La Terre : 15
La Lune : 4
Mars : 10.32
.
Toutes ces planètes seront sur le même plan, et graviteront autour du soleil, la lune gravitant elle autour de la terre. La réalisation d'un système complet utilise des notions mathématiques et astronomiques qui dépassent largement le cadre de ce TP.
Déterminer les orbites (primitives OpenGL) pour avoir un rendu "réaliste"

Le travail à réaliser se décomposer en trois parties :

Partie I:

- Mise en place des objets planètes(le soleil, mercure, vénus, la terre et la lune) et de leurs orbites.
- Réaliser l'animation en choisissant l'une techniques disponibles avec OpenGL (ProjetAnimation).
- Coder pour l'interface clavier les touches suivantes:

- 'r' : Lance/Arrête de l'animation
- '-' : Zoom arrière.
- '+' : Zoom avant.

- 'x' rotation/axe des x
- 'y' rotation/axe des y
- 'z' rotation/axe des z
- 'X' translation suivant l'axe des x
- 'Y' translation suivant l'axe des y
- 'Y' translation suivant l'axe des z
- 'o' afficher/cacher les orbites.'
- 'l' affiche la scène avec une caméra embarquée sur la lune orientée vers la terre.

Partie II:

- Finalisation du système avec l'ajout de lumières.
-

Partie III: Pour ceux qui ont déjà fini:

- Essayer de mapper des textures (voir l'exemple fourni).

## Anti-aliasing

L'OpenGL est une « state machine » : les paramètres de dessin consistent en une série de variables (très nombreuses, voir les annexes de la spécification OpenGL pour une liste complète).

La plupart sont modifiables par :

***void glEnable( GLenum cap )***

***void glDisable( GLenum cap )***

Pour l'antialiasing, les paramètres sont : GL_POINT_SMOOTH, GL_LINE_SMOOTH, et GL_POLYGON_SMOOTH. Le but de l'antialiasing est de rendre apparemment plus lisse les lignes en faisant un dégradé sur les contours des points.

---

Sites internet consacrés à l'OpenGL:

[1] : http://www.opengl.org

[2] : http://www.gamedev.net/nehe

[3] : http://www.antoche.fr.st

[4] : http://www.openuniverse.org