Qt / OpenGL

et la baffothérapie

- Travaux pratiques – S.T.S. I.R.I.S. 2ème année -

Temps alloué (sur poste) : **8 + 8 heures**

Pré-requis	• Programmation en langage ANSI C/C++ • Concepts de Programmation Orientée Objets, • Méthode de développement d'application Trolltech Qt version 4.x • Mise en œuvre des outils de développement sous Linux, Windows, … • Programmation OpenGL
Objectifs	• Programmation événementielle • Fabrication d'une librairie partageable • Exploitation d'une bibliothèque de classes C++/OpenGL, scène 3D
Critères d'évaluation	• Respect du Cahier des Charges et de la démarche indiquée • Respect des standards de codage en vigueur • Qualité et pertinence du compte-rendu

1. Installation de la librairie QtGLam

Récupérer l'archive `QtGLam-0.3x-dev.tar.gz` (*ou version supérieure*) et le paquetage de la documentation HTML attenante.

Décompresser l'archive ; elle contient un certain nombre de classes C++ dédiées à la mise en œuvre de scènes OpenGL, ici sous environnement Qt...
Étudier le contenu du fichier `README` afin d'installer la bibliothèque.

Tester le programme de démonstration fourni afin de valider l'installation.

Créer un répertoire de travail pour les travaux décrits dans ce document.

2. Validation de la chaîne de développement

La collection QtGLam contient une classe nommée `GLamWidget`. Cette classe propose un modèle de développement sous environnement Qt et offre un certain nombre de fonctionnalités de manipulations de la scène 3D...

Créer une application Qt minimale dont l'unique *widget* de la fenêtre d'application est une instance de `GLamWidget`. Cette application est constituée au minimum d'un point d'entrée (`main.cpp`) et d'une surcharge de `QMainWindow`... sans oublier l'indispensable fichier projet !

Pour ceux qui auraient oublié comment on fait, tournez la page...

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

class GLamWidget ;

class MainWindow : public QMainWindow
{
      Q_OBJECT

  public:
      MainWindow() ;

  private:
      GLamWidget*  glamWidget ;
} ;

#endif
```

mainwindow.cpp

```
#include <QtGui>
#include "mainwindow.h"
#include <qt4/GLam/glamwidget.h>

MainWindow::MainWindow()
      : glamWidget(NULL)
{
      glamWidget = new GLamWidget(this) ;
      setCentralWidget( glamWidget ) ;
}
```

main.cpp

```
#include <QApplication>
#include "mainwindow.h"

int main(int argc, char **argv )
{
      QApplication app(argc, argv ) ;

      MainWindow mainWin ;
      mainWin.resize( 400, 400 ) ;
      mainWin.show() ;

      return app.exec() ;
}
```

test.pro

```
HEADERS     += mainwindow.h

SOURCES     += main.cpp \
               mainwindow.cpp

UI_DIR       = .uic
RCC_DIR      = .uic
MOC_DIR      = .moc
OBJECTS_DIR  = .obj

TEMPLATE     = app
QT          += opengl
CONFIG      += warn_on release

DEFINES     += QTGLAM
QMAKE_LIBS  += -lQtGLam
```

Générer le programme exécutable. Il doit faire apparaître une grille 16x16 et un trièdre 0XYZ en bas à gauche. Tester les actions offertes par la classe `GLamWidget` (rotations, zoom, animation, …).

Si tout va bien, il est grand temps de jeter un coup d'œil à la documentation de QtGLam …

3. Création d'une classe d'application

Afin de développer maintenant notre propre application Qt/OpenGL tout en bénéficiant des possibilités offertes par la classe `GLamWidget`, commençons par spécialiser (ou dériver) cette dernière...

Créer une nouvelle classe dérivée de `GLamWidget`. Cette nouvelle classe doit posséder au minimum les membres suivants :

- un constructeur,
- un destructeur,
- une surcharge de chacune des méthodes protégées de la classe de base : `createObjects()`, `drawObjects()`, `animateObjects()` et `keyPressEvent()`.

Si la visualisation de la grille et du trièdre par défaut proposés par `GLamWidget` est souhaitée, la surcharge de `createObjects()` doit invoquer en premier lieu son homonyme parent par l'instruction `GLamWidget::createObjects();` et bien entendu, il faut appliquer la même démarche pour la routine d'affichage `drawObjects()`...

Le fait de surcharger `animateObjects()` ne remet pas en cause l'animation proposée par défaut par la classe de base, mais elle permettra le moment venu d'ajouter nos propres animations...

Enfin, l'interception des événements en provenance du clavier autorisera la mise en place de nos propres commandes... Si les commandes traitées par la classe de base doivent être conservées, il convient de lui retransmettre l'événement en fin de méthode.

Adapter la classe `MainWindow` et le fichier projet afin que le *widget* unique et principal de la fenêtre d'application soit maintenant une instance de la nouvelle classe...

Vérifier que tout va toujours bien ! Si la visualisation de la grille et du trièdre a été conservée, ils doivent apparaître ! Sinon, il doit au moins y avoir une fenêtre vide aux bonnes dimensions initiales, avec un fond blanc...

4. Technique d'ajout d'un objet sur la scène

Rien de plus simple ! Prenons l'exemple d'un parallélépipède et d'un cylindre...

- Ajouter des attributs privés :

```
GLamCube*     cube ;
GLamCylinder* cyl ;
```

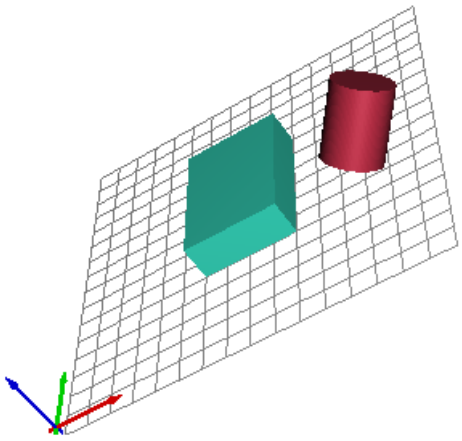
- Créer les objets dans `createObjects()` :

```
cube = new GLamCube( 4, 6, 1.5 ) ;
cube->setColor( 0.2, 0.8, 0.7 ) ;

cyl = new GLamCylinder( 2.4, 3, 90 ) ;
cyl->setColor( 0.8, 0.2, 0.3, true ) ;
```

- Provoquer le dessin des objets dans `drawObjects()` :

```
glPushMatrix() ;
      cube->draw() ;
      glTranslated( 5, -2, 1 ) ;
      glRotated( 60, 0, 1, 1 ) ;
      cyl->draw() ;
glPopMatrix() ;
```



- Détruire les objets dans le destructeur de la classe :

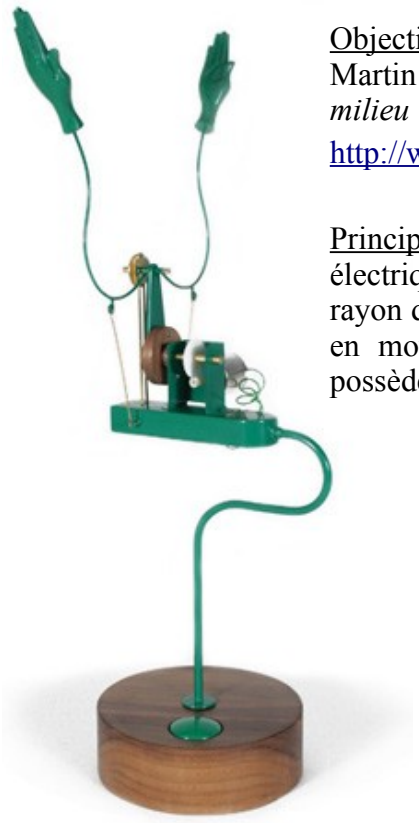
```
delete cube ;
delete cyl ;
```

Il est ainsi possible de tester tous les types d'objets proposés par la librairie GLam en s'inspirant des exemples fournis dans la documentation...

Attention : Pour les travaux qui suivent, il est conseillé de placer une instruction de changement d'échelle `glScaled(0.1,0.1,0.1)` au début de la méthode `drawObjects()`.

Maintenant que la chaine de développement est validée et la boîte à outils à portée de main, il est grand temps de s'attaquer à de vrais problèmes de modélisation 3D...

5. Modélisation 3D



Objectif : Modélisation de la « machine à applaudir » du designer anglais Martin Smith (*qui peut devenir machine à baffes si l'on met la tête au milieu !*).

http://www.smithautomata.co.uk

Principe général : Un interrupteur situé sur la base permet d'alimenter électriquement ou non un moto-réducteur. Celui-ci entraîne une roue sur un rayon de laquelle est articulée une bielle qui transforme le mouvement rotatif en mouvement vertical linéaire alternatif. L'autre extrémité de la bielle possède un axe qui actionne les deux bras...

Simplification du modèle :

Tant pis pour le coté artistique, nous allons quelque peu simplifier la machine dans sa version modèle 3D en limitant au maximum le nombre de pièces !

(à voir en annexe, une suggestion de dessin de définition).

5.1. Modélisation du moteur : Celui-ci peut être représenté dans un premier temps sous forme d'un simple parallélépipède et d'un cylindre pour son axe.

5.2. Modélisation du train d'entrainement : Cet ensemble comprend les deux roues dentées, l'axe principal (6) et le volant d'entrainement (8). Les roues dentées ont un module m égal à 1, l'entraxe est donné par la formule $2m.(n_1 + n_2)$ où n_1 et n_2 sont le nombre de dents de chacune des roues. Le rapport de transmission est égal à n_1/n_2.

(la documentation de la librairie GLam donne un exemple de définition de deux roues dentées correctement engrenées...).

5.3. Modélisation de la bielle : Celle-ci peut être représentée dans un premier temps dans sa version la plus simple. Ne pas oublier de matérialiser l'axe radial du volant repéré (9), et l'axe guide (11).

5.4. Modélisation du bâti : Il est constitué du support (1), du buste (2) et de l'interrupteur M/A (12). Les évidements pratiqués dans le buste, en dehors du guide pour l'axe (11) et du trou pour l'axe (6) pourront être traités plus tard, ils permettront de mieux observer les pièces en mouvement qu'elle que soit le point de vue.

5.5 Assemblage de l'ensemble : Si ce n'est déjà fait, positionner les différents ensembles réalisés à leurs position et orientation respectives sur la scène en respectant l'indication d'origine de repère présentée sur le dessin (machine en élévation sur l'axe Z, plan 0XY au niveau de la base du moteur).

La bielle doit être dessinée en position haute au démarrage (comme sur le dessin de définition).

Prévoir dès maintenant le paramétrage de l'animation en maintenant l'angle de rotation de l'axe moteur dans un attribut privé. Ce paramètre permettra de proche en proche le calcul de la position et de l'orientation du volant, de la bielle, de l'axe guide et des futurs ensembles bras/main...

6. Animation 3D

Il s'agit maintenant d'animer notre superbe machine ! Bien évidemment, pour un rendu réaliste, les mouvements de l'ensemble des pièces mobiles doivent être synchronisés.

6.1. Bouton Marche/Arrêt : En s'inspirant du code de `keyPressEvent()` de la classe GLamWidget, détecter l'appui sur la barre d'espace afin de mettre à jour un attribut privé booléen maintenant l'état du système (en marche ou à l'arrêt). Modifier la couleur du bouton (12) en fonction de l'état : vert pour Arrêt (défaut), ou rouge pour Marche.

6.2. Démarrage du moteur : Implémenter la surcharge de méthode `animate()` afin de faire varier l'angle instantané de l'axe moteur lorsque la machine est en marche. L'angle est en degrés, la progression peut être du style `angle += k` avec k = 1 par défaut.

L'exemple GLamGear de la documentation montre comment mettre les éléments en rotation en fonction de la valeur d'angle.

En option, on peut prévoir de modifier K par pas de 0.5 de manière dynamique, par exemple à partir des touches + et – du clavier...

6.3. Mouvement de la bielle : Faire un petit croquis afin de calculer la position et l'inclinaison de la bielle en fonction de l'angle du volant... Bon d'accord, je vous aide ! mais faites le croquis quand même pour vérifier !

D : distance entre l'axe de (6) et l'axe de (11)
R : rayon entre l'axe de (6) et l'axe de (9)
L : longueur de la bielle (entre ses axes)
α : angle de R p/r à l'horizontale en radians
β : angle de la bielle p/r à la verticale en radians

$$D = R.\sin(\alpha) + \sqrt{L^2 - (R.\cos(\alpha))^2}$$

$$\beta = \arcsin\left(\frac{R.\cos(\alpha)}{L}\right)$$

Compléter le code afin de réaliser l'animation synchronisée de l'ensemble... *beautiful, isn't it ?!*

7. Finalisation du modèle

7.1. Pièces manquantes : Définir les éléments manquants « bras/main » (sur papier ou avec un outil type OpenOffice Draw, sans oublier la cotation), et ajouter au programme le code nécessaire afin de compléter le modèle...

La main peut être représentée sous forme d'une simple plaque. Les bras sont articulés autour de deux axes fixés sur le buste, ils sont pilotés par l'axe (11) grâce à une sorte de fourche (un simple trou cylindrique coincerait toute la machine...).

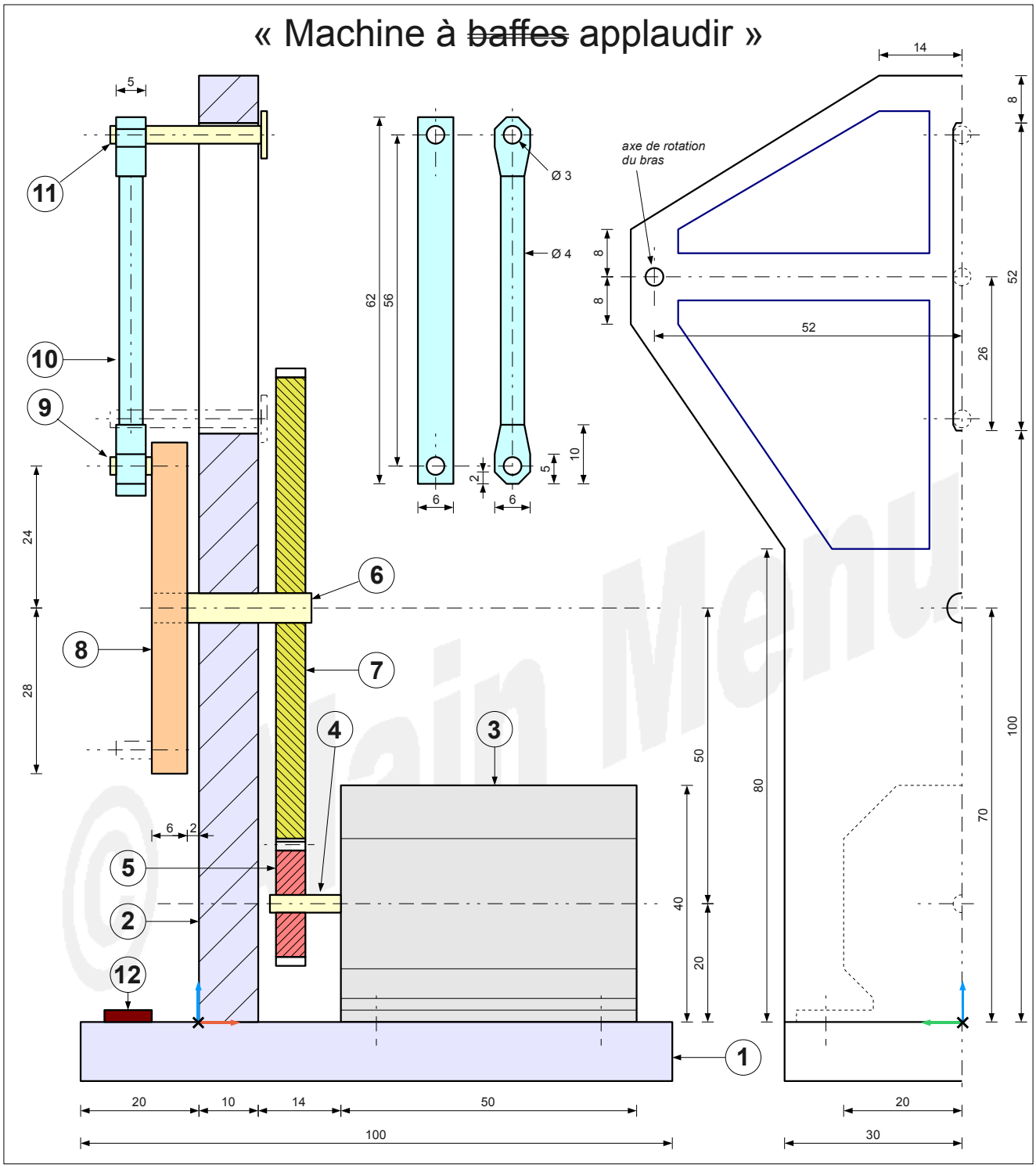
Note: L'exemple « pince » de la documentation GLam contient un élément nommé *bras1* basé sur le même principe.

7.2. Amélioration du rendu : Le cas échéant, améliorer la représentation de la bielle en implémentant la deuxième version proposée, et ajouter les évidements du buste afin de mettre en valeur le mécanisme en action.

Parfaire le rendu global de la machine en attribuant des couleurs et/ou des matériaux à chaque élément (*l'original est disponible en six coloris !*).

Et pour finir ? Contactez une galerie d'Art Moderne et négociez une expo !

Annexe :



note : les bras et mains ne sont pas représentés...

Nomenclature

1	support 100 x 60 x 10	7	roue dentée : module 1, 80 dents
2	buste	8	volant Ø56 x 6
3	bloc moteur	9	axe de bielle Ø3 x 13
4	axe moteur Ø3 x 12	10	bielle 62 x 6 x 5 (*deux versions...*)
5	roue dentée : module 1, 20 dents	11	axe guide de translation Ø3 x 25.5 + tête plate Ø8
6	axe du train d'entrainement Ø5 x 27	12	interrupteur Marche/Arrêt