L’infographie

Introduction à Open GL

Références

- Ce tutoriel est inspiré des tutoriels en ligne de :
  - Nicole Deflaux Terry (Tulane University),
  - Matthew Suderman (McGill University).
- et des ouvrages :
  - James D. Foley, Andries van Dam, Steven K. Feiner and John F. Hughes. Computer Graphics: Principles and Practice in C, 2nd edition. Addison Wesley, 1995.
  - Francis S. Hill, Jr., Computer Graphics Using Open GL, Second Edition, Prentice Hall, 2000.
  - Mason Woo, Jackie Neider, Tom Davis, OpenGL Programming Guide: The *Official Guide to Learning OpenGL, Version 1.2*, Addison Wesley, 1999.

Introduction à Open GL

- OpenGL est une **librairie graphique de bas niveau** pour tracer des objets géométriques en dimension 2 ou 3.
- Les objets complexes sont tracés à partir de combinaisons d´objets élémentaires.
- OpenGL est **indépendant du système et du langage** avec lesquels on l´utilise.
- OpenGL se lie avec les langages C ou C++ et avec d´autres langages tels que Java (package Java3D) ou Tcl.
- OpenGL n´a pas de capacité de fenêtrage.
- OpenGL est une **machine à états finis**.
- À tout instant, on connaît les paramètres graphiques courants du système (transformation, lumière...). Ils sont utilisés pour le tracé courant.

Le pipeline graphique

- **Les étapes du pipeline de rendu graphique**
  - Définition du **modèle mathématique** d´un objet graphique
  - Définition d´un **point de vue** de la scène graphique
  - Calcul des **couleurs et des lumières**
  - **Rasterisation** (transformation en points lumineux)

- Les exemples donnés dans ce cours reposent sur une programmation en C/C++ et utilisent un code basé sur la librairie GLUT pour la gestion des fenêtres et des événements. Les MFC peuvent aussi être utilisées.

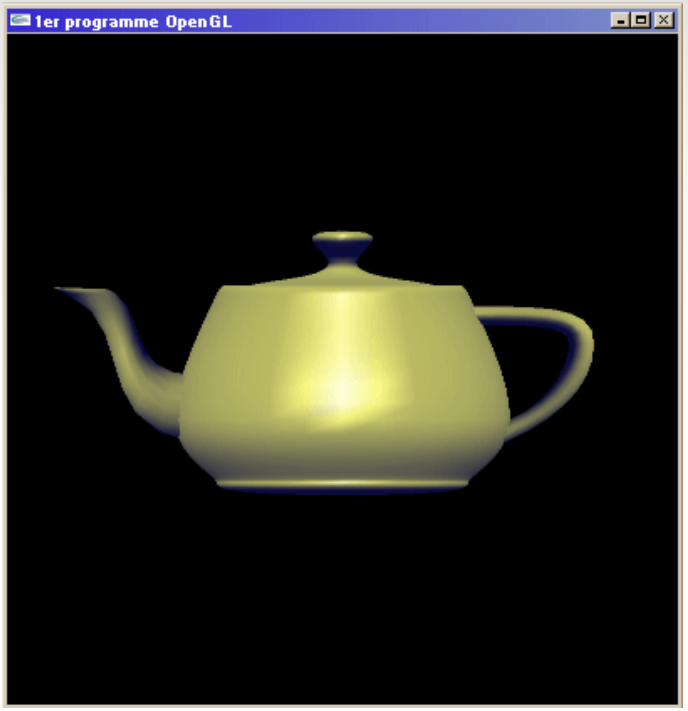
Le pipeline graphique (2)

- **Fonctionnement du pipeline graphique**
  - Avant d'être tracé, un point passe par :
    - **modèle de vue** (définition mathématique des objets),
    - **projection** (point de visualisation de la scène),
  - **port de vue** (application à une zone d´écran).
  - En cas de redimensionnement de la fenêtre graphique, les transformations de modèle de vue et de projection sont redéfinies :
    - Le **port de vue** est modifié pour que la scène représente toujours la même fraction de la fenêtre,
    - la **projection** est modifiée pour compenser la modification de la taille du port de vue,
      le volume de projection doit être changé dans son *rapport largeur sur hauteur* pour conserver les proportions des objets géométriques.

Exemple de programme C++/OpenGL

source : teapot.c
exécutable : teapot.exe

Pour compiler un programme utilisant OpenGL, il faut indiquer au compilateur l'emplacement des fichiers d'include gl.h, glu.h et glut.h et les librairies Opengl32, Glu32 et Glut32



La ligne de commande pour la compilation de teapot est donc (en c++ unix ou mingw windows) :

g++ teapot.c –o teapot –Iinclude –Llib –lopengl32 –lglu32 –l glut32

dépend du compilateur et du système

Analyse du programme d'exemple

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
```

fichiers d'inclusion

```
static void init(void) {
}

void display(void) {
}

void reshape(int w, int h) {
}

void idle(void) {
}

void keyboard(unsigned char key, int x, int y) {
}
```

fonctions pour OpenGL

```
int main(int argc, char **argv)
{
  glutInitWindowSize(400, 400);
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
  glutCreateWindow("1er programme OpenGL");
  init();
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutDisplayFunc(display);
  glutIdleFunc(idle);
  glutMainLoop();
  return 0;
}
```

main()

Analyse du *main()*

```
int main(int argc, char **argv)
{
  glutInitWindowSize(400, 400);              //déclaration d'une fenêtre
  glutInit(&argc, argv);             //initialisation de la fenêtre
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH); //type de contenu graphique
  glutCreateWindow("1er programme OpenGL");      //création de la fenêtre
  init();                          //initialisation de la scène OpenGL
  glutReshapeFunc(reshape);            //quelle fonction réaffiche quand on modifie la fenêtre
  glutKeyboardFunc(keyboard);                //quelle fonction pour l'interaction clavier ?
  glutDisplayFunc(display);            //quelle fonction affiche la scène
  glutIdleFunc(idle);                //que faire si on a du temps libre ?
  glutMainLoop();            //boucle sur le programme principal, en attente d'événements
  return 0;
}
```

Analyse de la partie OpenGL

```
/* initialisation d'OpenGL*/
static void init(void) {
   glShadeModel(GL_SMOOTH);
   glClearColor(0.0, 0.0, 0.0, 0.0);      //couleur de fond

    // Z Buffer pour la suppression des parties cachées
  glEnable(GL_DEPTH_TEST);

   glEnable(GL_LIGHTING);                  //utilisation d'une lumière
   glEnable(GL_LIGHT0);                |
                                       |
   GLfloat ambient[] = {0.0, 0.0, 1.0, 1.0};          |
   GLfloat diffuse[] = {0.9,0.9, 0.2, 1.0};           |
   GLfloat position[] = {0.0, 3.0, 3.0, -3.0};        |
                                       |
  glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);  |
  glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);      |
  glLightfv(GL_LIGHT0, GL_POSITION, position);  |
}
```

Analyse de la partie OpenGL

```
/* Vide la fenêtre et dessine la théiere */
void display(void) {
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  //effacement
   glColor3f (0.1, 0.1, 1.0);

GLfloat specular [] = { 1.0, 1.0, 0.5, 1.0 };          // definit les propriétés matérielles de la couleur spéculaire
GLfloat shininess [] = { 100.0 };              //et du degré de brillance.
glMaterialfv(GL_FRONT, GL_SPECULAR, specular);
glMaterialfv(GL_FRONT, GL_SHININESS, shininess);

glMatrixMode(GL_MODELVIEW);                    //définition des transformations sur la théière (rotation)
glLoadIdentity();
glRotatef(angle,0,1,0);

glutSolidTeapot(1.0);                          //mise dans la scène d'un objet
glFlush();                                 //vidage du pipeline (envoi au processeur vidéo)
glutSwapBuffers();
}
```

Analyse de la partie OpenGL

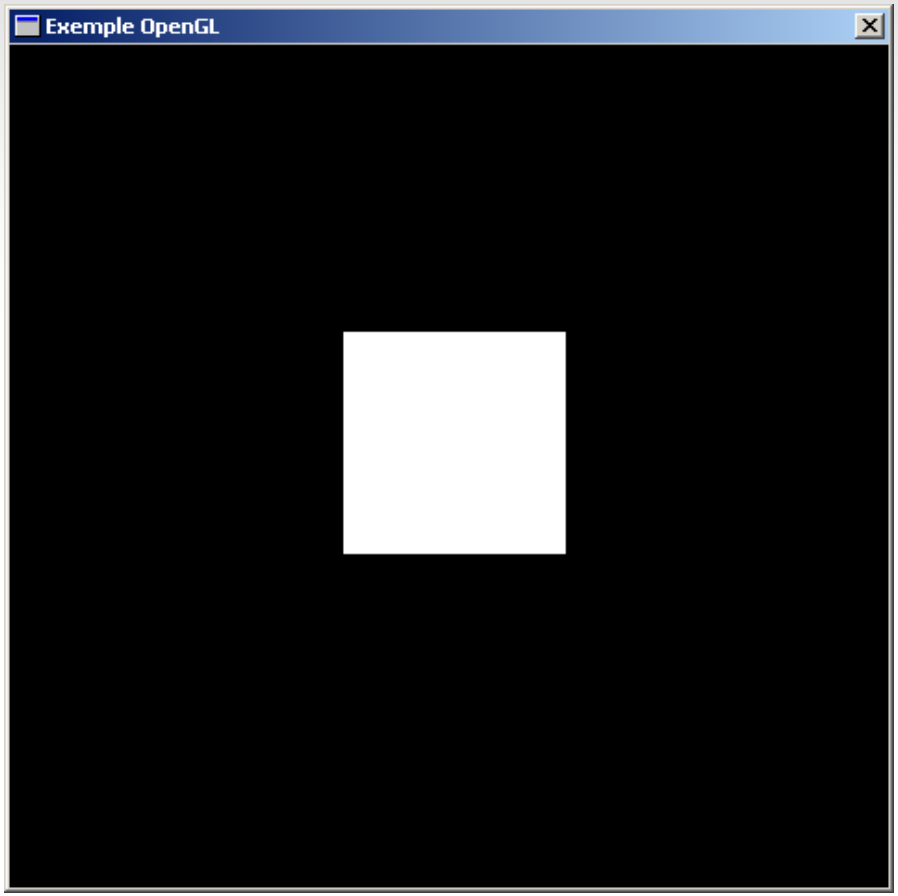
```
/* Au cas ou la fenetre est modifiee ou deplacee */
void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);    //définition de la zone de dessin

   glMatrixMode(GL_PROJECTION);                   //matrice de projection
   glLoadIdentity();
   glOrtho(-2, 2, -2, 2, -2, 2);

   glMatrixMode(GL_MODELVIEW);                    //matrice de transformation
   glLoadIdentity();
}
```

Premier exemple 1/5

- Tracé d’un rectangle noir sur fond blanc



Premier exemple 2/5

- Analyse du code
  - **Création de la fenêtre graphique**
    - Définition de la taille de la fenêtre et de sa position. Création de la fenêtre en lui attribuant un titre.

```
glutInitWindowSize(WIDTH, HEIGHT);
glutInitWindowPosition(0, 0);
glutCreateWindow("Premier exemple : carré");
```

  - **Définition des fonctions de call-back**
    - **Ce sont des fonctions qui sont automatiquement appelées lorsque les événements correspondants sont interceptés :**
      - **window_display est appelée chaque fois que l´on doit redessiner la fenêtre,**
      - **window_reshape est appelée chaque fois que l´on redimensionne la fenêtre,**
      - **window_key est appelée chaque fois qu´une touche clavier est pressée.**

Premier Exemple 3/5

- **Analyse du code (suite)**
  - **Tracé graphique**
    - À l´initialisation, on **définit la couleur du fond** qui sera utilisée lorsqu'on effacera le fond par **glClear(GL_COLOR_BUFFER_BIT)**.
    - glClearColor(RED, GREEN, BLUE, ALPHA); } Le tracé consiste à définir une **couleur** (le blanc) et une **forme** :
      un polygone à quatre sommets dont les sommets sont énumérés par **glVertex3f**.

```
glColor3f(1, 1, 1);
glBegin(GL_POLYGON);
    glVertex3f(-1, -1, 0);
    glVertex3f(1, -1, 0);
    glVertex3f(1, 1, 0);
    glVertex3f(-1, 1, 0);
glEnd();
```

  - Enfin, on indique à OpenGL de tracer la scène graphique qui vient d'être définie.

```
glFlush();
```

Premier Exemple 5/5

- **Redimensionnement de la fenêtre graphique**
  - glViewport(0, 0, width, height);
  - glMatrixMode(GL_PROJECTION);
  - glLoadIdentity();
  - glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);

  // toutes les transformations suivantes s´appliquent au modèle de vue
  - glMatrixMode(GL_MODELVIEW); }

Syntaxe des commandes OpenGL 1/4

- Toutes les commandes OpenGL commencent par **gl**.
  Les objets sont généralement encadrés par un **glBegin()** (avec une constante en **GL_...**) et un **glEnd()**.

```
glColor3f(1, 1, 1);
glBegin(GL_POLYGON);
    glVertex3f(-1, -1, 0);
    glVertex3f(1, -1, 0);
    glVertex3f(1, 1, 0);
    glVertex3f(-1, 1, 0);
glEnd();
```

- Il est recommandé d'**indenter** entre un **glBegin()** et un **glEnd()**.

Syntaxe des commandes OpenGL 2/4

- **Flexibilité sur les types des arguments**
- Certaines noms de commandes OpenGL ont de 1 à 3 lettres finales qui dénotent le nombre et le type de leurs paramètres.
- Premier caractère : **nombre de paramètres** (**2**, **3**, ou **4**),
- deuxième (et troisième éventuellement) caractère : **type des arguments**
  - **f** : float
  - **d** : double float
  - **s** : signed short integer
  - **i** : signed integer
  - **b** : character
  - **ub** : unsigned character
  - **us** : unsigned short integer
  - **ui** : unsigned integer
- dernier caractère (**v**) : argument sous forme d´un **vecteur** au lieu d´une **liste de scalaires**.

Syntaxe des commandes OpenGL 3/4

- **Cas de la commande glColor*()**
- Chaque commande OpenGL a une documentation associée. Vous pouvez regarder dans le redbook (livre de référence) ou directement sur le site http://www.opengl.org/
- Lorsqu'on nomme une **commande typée**, on omet les lettres de typage en les remplaçant par une *****.
- par exemple **glColor*** désigne glColor3b, glColor3d, glColor3f, glColor3i, glColor3s, glColor3ub, glColor3ui, glColor3us, glColor4b, glColor4d, glColor4f, glColor4i, glColor4s, glColor4ub, glColor4ui, glColor4us, glColor3bv, glColor3dv, glColor3fv, glColor3iv, glColor3sv, glColor3ubv, glColor3uiv, glColor3usv, glColor4bv, glColor4dv, glColor4fv, glColor4iv, glColor4sv, glColor4ubv, glColor4uiv, glColor4usv

Syntaxe des commandes OpenGL 4/4

- **Trois exemples**
- On a la **même sélection de couleur** (le blanc opaque) avec les trois commandes suivantes :
  - / / 3 floats

    glColor3f(1.0, 1.0, 1.0);
  - // 3 unsigned bytes

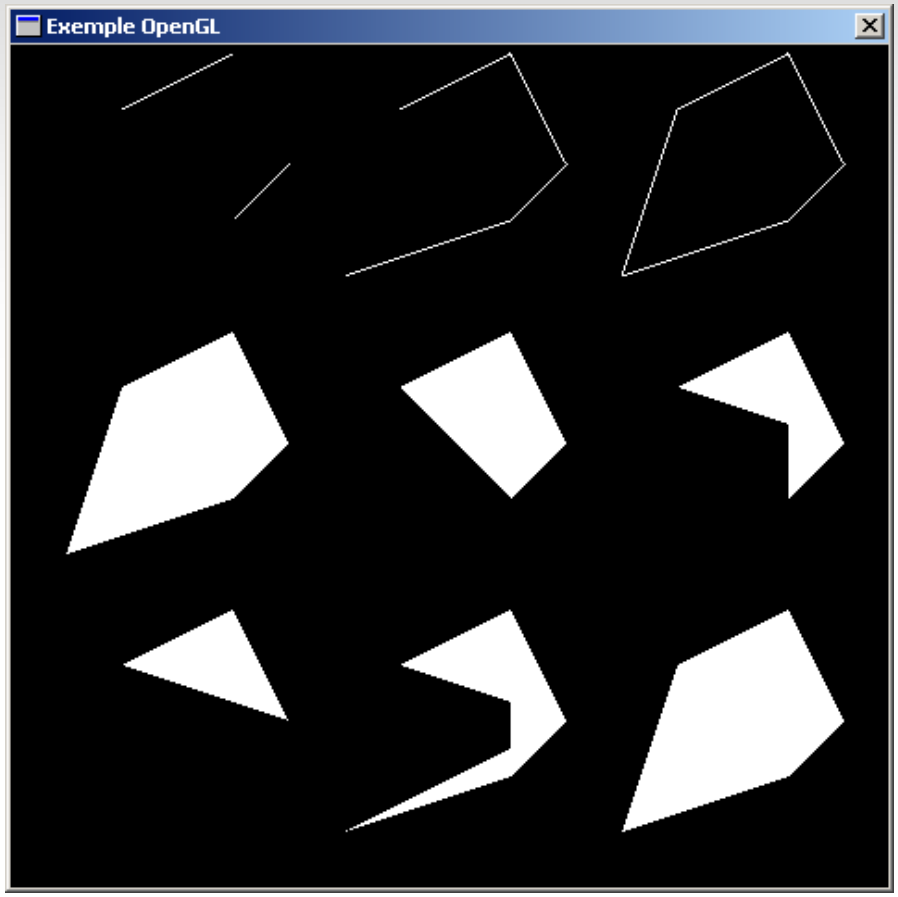
    glColor3ub(255, 255, 255);
  - // 4 floats sous forme de vecteur

    double vecteur[4] = {1.0, 1.0, 1.0, 1.0};

    glColor4dv(v);

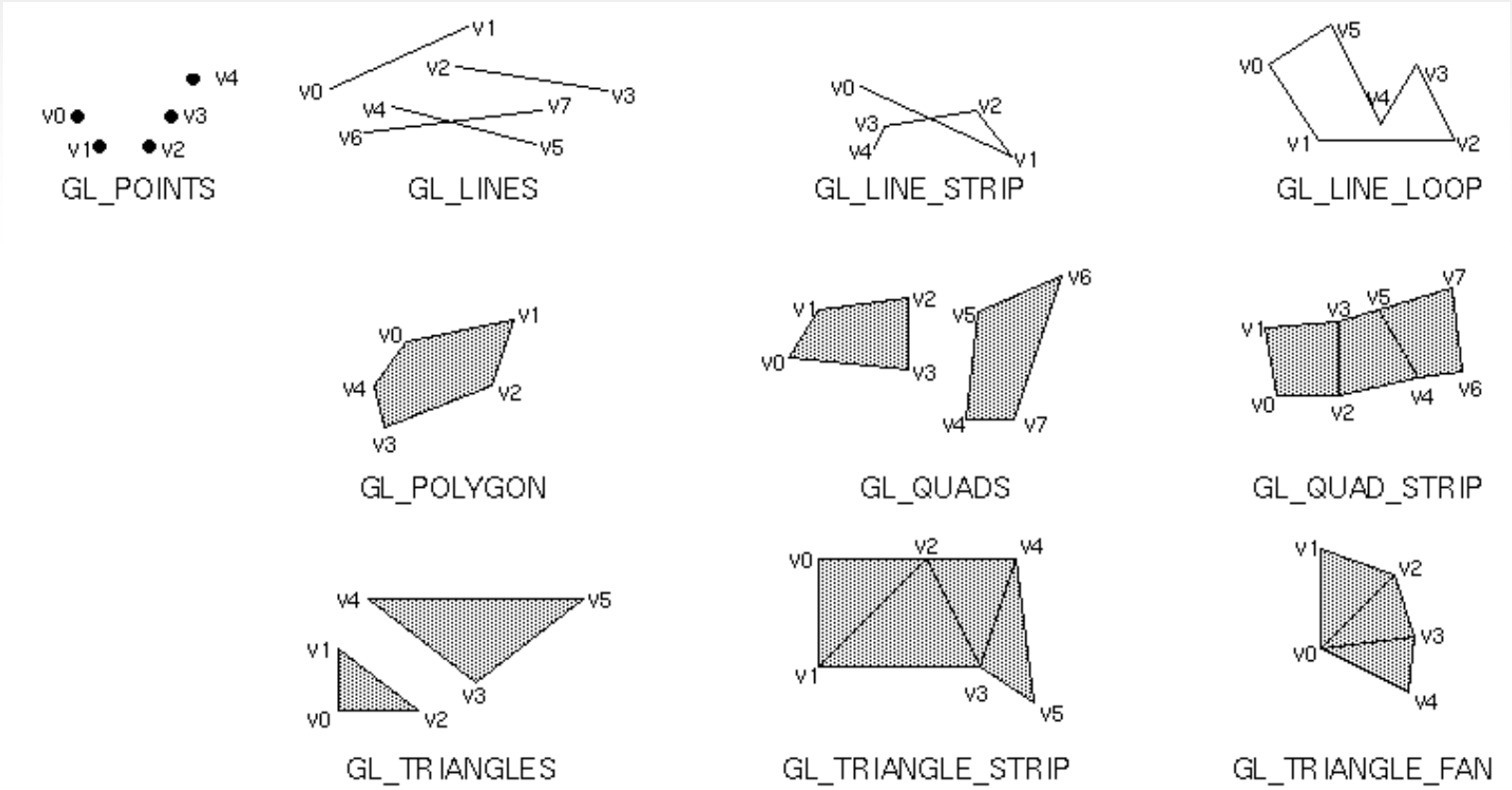
Les primitives 1/3

- **Tracé de primitives géométriques**
- Ces figures sont obtenues avec le programme Primitives.cpp (voir cours N°2)



Les primitives 2/3

- Les primitives suivantes sont disponibles aux programmeurs en OpenGL.



Les primitives 3/3

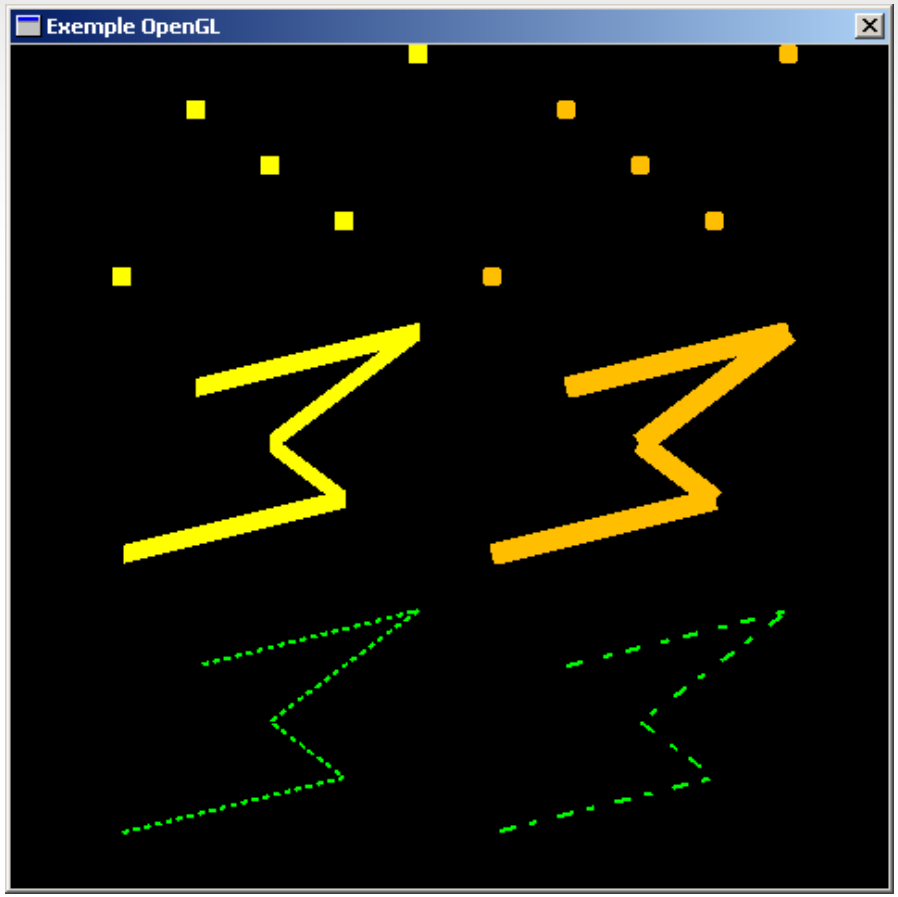
- Toute primitive se trace au moyen d´une suite de sommets donnés par **glVertex** et situés entre un **glBegin()** et un **glEnd()**.
- // figure en bas à droite dans la diapo précédente

```
glBegin(GL_TRIANGLE_FAN);
    glVertex2f(-1.0, 1.0);
    glVertex2f(1.0, 2.0);
    glVertex2f(2.0, 0.0);
    glVertex2f(1.0, -1.0);
    glVertex2f(-2.0, -2.0);
glEnd();
```

- Le **tracé effectif** d´une primitive n´a lieu que lors de la rencontre d´une commande **glFlush();**

Les variables d’état 1/6

- **Tracé de primitives géométriques : possibilité de changer le style et la couleur des primitives**



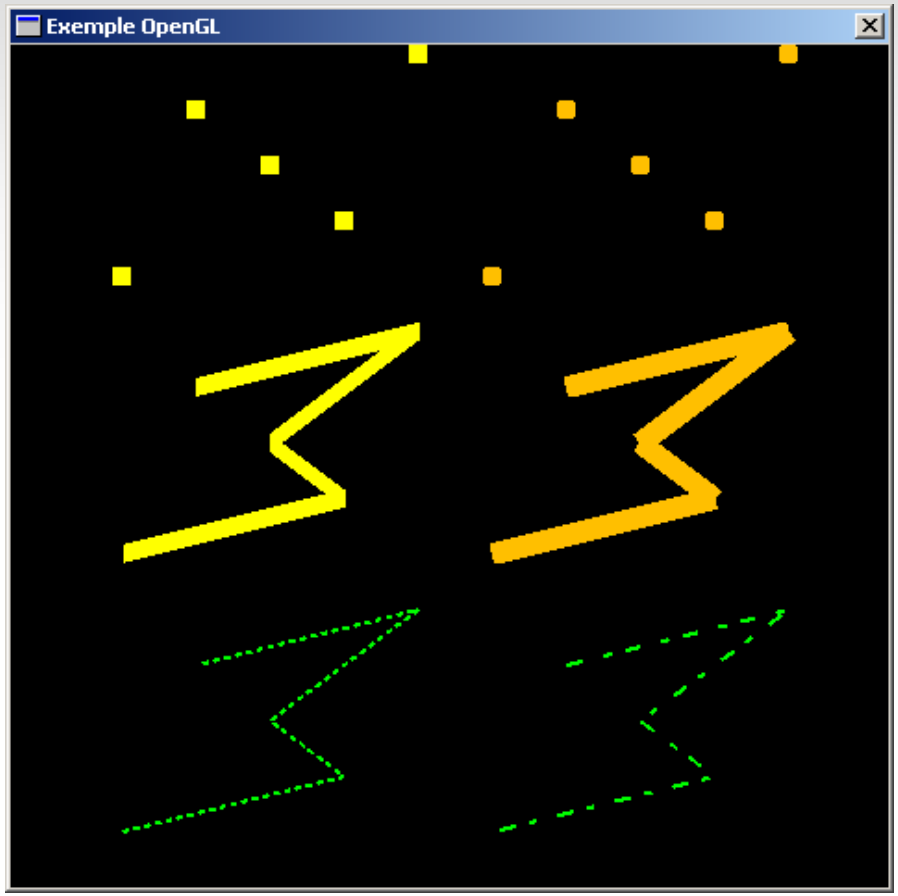
Les variables d’état 1/5

- Le tracé graphique est affecté par la valeur des **variables d´état** qui constituent l'**environnement graphique** courant.
- **Variables de mode**
  - Il existe deux modes d´ombrage définies par la commande **glShadeModel** qui modifie la variable d´état de mode **GL_SHADE_MODEL** :
    - glShadeModel(GL_SMOOTH) : **ombrage doux**,
    - glShadeModel(GL_FLAT) : **ombrage plat** (la valeur par défaut).

Les variables d'état 2/5

- Les variables booléennes sont activées et désactivées au moyen des commandes **glEnable()** et **glDisable()**.
  - **GL_POINT_SMOOTH** : tracé des points avec le filtrage correct ou antialiasés,
  - **GL_LINE_SMOOTH** : tracé des droites avec le filtrage correct ou antialiasés,
  - **GL_LINE_STIPPLE** : tracé en pointillés pour les droites.
- **Exemple**

```
// Dessine les points filtrés //
    Exemple en haut à droite
    glEnable(GL_POINT_SMOOTH)
    ;
```



VARIABLES D'ÉTAT 3/5

- **Variables d´état valuées (fin)**
  - **Exemple de ligne pointillée**

```
// Active le pointillé
glEnable(GL_LINE_STIPPLE);
// Définit le patron de pointillés 1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 0
glLineStipple(2, 0x0C0F);
glBegin(GL_LINES);
   glVertex2f(-2.0, 0.0);
   glVertex2f(2.0, 0.0);
glEnd();
```

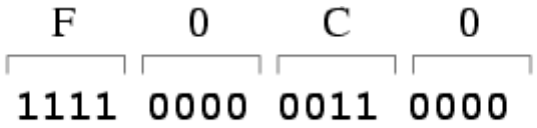
Les variables d'état 4/5

- **Autres variables valuées**
  - **Taille des points** : **glPointSize(GLfloat *taille*)** où *taille* est la taille en pixel souhaitée pour le tracé des points.
  - **Largeur des lignes** : **glLineWidth(GLfloat *largeur*)** où *largeur* est la largeur en pixel souhaitée pour le tracé des lignes.
  - **Lignes pointillées** : **glLineStipple(GLint *facteur*, GLushort *patron*)** où *patron* est une série de 16 bits à 0 (pas de tracé) ou 1 (tracé).
    Ce patron est répété autant que nécessaire pour le tracé d´une ligne.
    Le paramètre *facteur*, entre 0 et 255, est un coefficient de dilatation du patron.
    GL_LINE_STIPPLE doit être activé par glEnable(**GL_LINE_STIPPLE**).

Les variables d’état 5/5

- **Exemple de ligne pointillée**

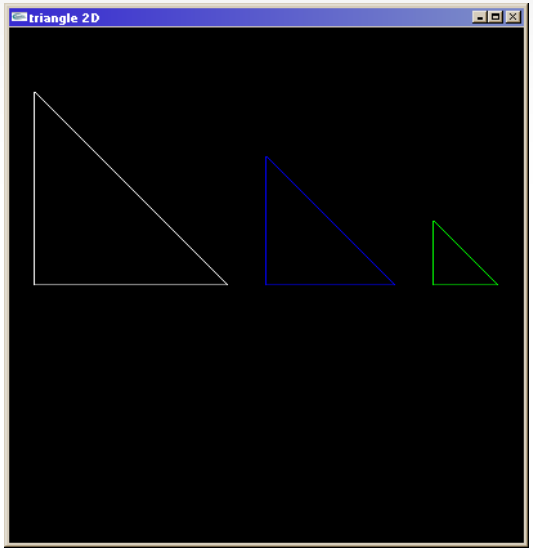
```
// Active le pointillé
glEnable(GL_LINE_STIPPLE);
// Définit le patron de pointillés 1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 0
glLineStipple(2, 0x0C0F);
glBegin(GL_LINES);
    glVertex2f(-2.0, 0.0);
    glVertex2f(2.0, 0.0);
glEnd();
```



A vous de jouer !

- Travail en 2D sur les LineLoop (polylignes fermées)
  - créer un triangle isocèle rectangle blanc, du type

  - créer trois triangles isocèles rectangles, de couleurs différentes (voir la fonction glColor() dans le RedBook), régulièrement espacés



- Cercle
  - créer un cercle (en paramètre : centre, rayon, nb points)
  - modifier la couleur du cercle
  - chercher dans le RedBook comment faire pour remplir ou, ne pas remplir le polygone

Utilisez le fichier d'exemple à modifier : carre.*cpp*