# Introduction à OpenGL

1er octobre 2009

# Généralités

- A l'origine la librairie GL a été développée en 1989 par Sillicon Graphics, puis portée sur d'autres architectures en 1993 (OpenGL)
- GL : Librairie graphique standard de Silicon Graphics (orientée visualisation). Utilisée sur les stations SGI ainsi que sur les stations d'autres constructeurs (sous licence).
- **OpenGL** : Sous-ensemble de GL ne nécessitant pas une licence SGI (orienté visualisation)
- GL et OpenGL sont des **librairies graphiques d'affichage 3D**, c'est à dire une interface logiciel permettant d'effectuer des opérations d'affichage sur un écran graphique

OpenGl est une norme en infographie. La librairie est utilisé dans de nombreux logiciels :

- Logiciels scientifiques
- Certain jeux vidéo sous windows : Call of Duty, Doom, Quake, ... La majorité des jeux 3D sous Linux
- Logicel necessitant une représentation en 3D : GoogleEarth, Second Life,...
- Conception 3D
- ect...

- OpenGl propose des versions pour la plupart des **systèmes d'exploitations** :
  - Unix
  - Linux
  - Iris
  - AIX-OS/2 (IBM)
  - Windows (Microsoft)
  - Solaris (Sun MicroSystems)
  - etc

  L'implementation libre d'OpenGL (sous licence X11) utilisé principalement sous Linux s'appel **MESA**
- OpenGl est exploité à partir des plusieurs **langages de développements** :
  - Ada
  - C, C++
  - Fortran
  - Pyton
  - Java
  - etc

- OpenGl est constitué d'environ **200 fonctions graphiques** qui permet de :
  - Définir des **objets** graphiques simples,
  - Définir des **couleurs**,
  - Définir les **points de vue**,
  - Définir les **sources de lumière**,
  - Etc
- OpenGl est une **machine a états** qui permet de définir un contexte de tracé :
  - Position de caméra
  - Projection
  - Couleurs
  - Lumières
  - Matériaux
  - etc

  Les états sont représentés par des **variables d'états**.

## Exemple

La **couleur** est une variable d'état. Une fois fixée, tous les objets seront dessinées dans cette couleur jusqu'à ce qu'on change cet état (c à d cette couleur).
D'autres variables d'états :

- le type de dessin en cours (points, lignes, polygones)
- le trait de dessin ( taille, motif)
- état de la lumière (allumée / éteinte, position, ...)

- OpenGL donne des **ordres de tracé** de primitives graphiques (facettes, etc) directement en 3D.
- OpenGL produit des **images synthétiques** sophistiquées en temps réel si on dispose du hardware adéquat
- OpenGl se charge de faire les **changements de repère, la projection à l'écran, le clipping, l'élimination des parties cachées, l'interpolation des couleurs et de rasterisation** (tracer ligne à ligne).
- OpenGL s'appuie sur le hardware disponible selon la carte graphique. Toutes les opérations de base sont a priori accessibles sur toute machine, simplement elles **iront plus ou moins** vite selon qu'elles sont implémentées au niveau matériel ou logiciel.

- OpenGl est un **API (Application Programmer Interface)**, pas un protocole. La spécification est **indépendante** du :
  - Système de fenêtrage
  - Système d'exploitation
- OpenGl ne comporte **aucune fonction de gestion d'écran** ou de périphériques tels que la souris ou le clavier. Ces opérations sont déléguées aux environnements d'accueil tels que XWindow, Glut, ou SDL

# Organisation d'ensemble : les 3 moteurs

La machine OpenGL (graphic engine) se décompose en plusieurs moteurs :

- **Le geometric engine**
- **Le raster engine**
- **Le raster manager**,

# Le geometric engine

Le geometric engine s'occupe de la partie purement 3D : triangulation des polygones, changements de repère, projection en perspective à l'écran, clipping (élagage de ce qui sort de l'écran).

# Le raster engine

Le raster engine prend en charge la partie 2D :

- il rasterise les triangles de manière à produire des pixels (que l'on nomme fragments tant qu'ils ne sont pas véritablement inscrits à l'écran),
- interpole au passage les couleurs et les coordonnées texture,
- puis élimine les parties cachées par Z-buffer : en fonction de la profondeur z du fragment en cours de tracé et du z actuellement stocké dans le pixel visé, le pixel est ou non modifié (i.e. sa couleur et son z).

Ceci constitue le schéma de base, on verra par la suite qu'OpenGL offre de nombreux mécanismes supplémentaires et moyens de contrôle.

# Le raster manager

Le raster manager a qui le pixel est confié pour être tracé à l'écran après d'éventuels traitements supplémentaires (dithering, etc).

# Bibliothèque OpenGL

## OpenGL Library (GL)

**Préfixe des fonctions : gl**

Librairie standard proposant les fonctions de base pour l'affichage en OpenGL.

## OpenGL Utility Library (GLU)

**Préfixe des fonctions : glu**

Surcouche à GL

Fournit des fonctions plus évoluées :

- transformations géométriques
- triangulation des polygones
- rendu des surfaces paramétriques et quadriques
- ...

# SDL et OpenGL

- OpenGL ne gére pas l'interface avec l'utilisateur : la cration d'une fenêtre, les périphérique tel que le clavier, la sourie ou un joystick.
- Il exite des libraire associées à OpenGL pour remplir ce role comme : AUX, GLX, GLUT, **SDL** ...
- SDL est spécialisé pour la création de jeux vidéo 2D comme 3D

# Utilisé OpenGL avec SDL

## Création d'un contexte OpenGL

- On doit créer un contexte pour OpenGL
- Pour cela on utilise le flag SDL_OPENGL, lors de l'appel de la fonction SDL_SetVideoMode
- Ex : SDL_SetVideoMode(640, 480, 8, SDL_OPENGL) ;

## Boucle d'évenement

- On doit actualiser la scène à interval regulier.
- Donc on utilisera SDL_PollEvent (non bloquante) (plutôt que SDL_WaitEvent, qui est bloquante) pour recuperrer les evenements.
- On ajoute dans la boucle la fonction SDL_Delay, pour éviter de trop utiliser les resources du CPU.

## Actualisation de la scène

- A chaque fois qu'on modifie la scène (redefinition d'objets,changement d'une variable d'état,...) on doit appeler la fonction SDL : SDL GL SwapBuffers( )

## Librairie et fichier d'en tête

- Pour utiliser les fonctions de la librairie d'OpenGL on ajoute le fichierd d'en tête **gl.h**
- On utilise la librairie libGL
- Compilation : **gcc -lGL -lGLU -lSDL -o executable sources.c**
- Voir la documentation sur le web : http ://www.libsdl.org/

# Exemple d'un programme SDL avec OpenGL

```
SDL_Surface *screen;
SDL_Event event;

/* Initialise la libraire SDL */
SDL_Init(SDL_INIT_VIDEO|SDL_INIT_TIMER);

/* Netoye au moment ou le programme quitte */
atexit(SDL_Quit);

/* Initialise l'affichage en 640x480 avec des pixel de 8 bits */
screen = SDL_SetVideoMode(640, 480, 8, SDL_OPENGL);

/* Initialisation des parametres OpenGl*/
init_gl();

/*Boucle infinie*/
while ( 1 ) {

        /*dessine la scene*/
        draw_scene(screen);

        /*Recupere les evenements (si y'en a un)*/
        while(SDL_PollEvent( &event ))
                /*Traite l'evenement*/
                process_event(&event);

        /*Attends 50 ms (~20 image/s)*/
        SDL_Delay(50);
}
```

# Syntaxe des commandes OpenGL

- Toutes les fonctions OpenGL sont composées d'un **préfixe** et éventuellement d'un **suffixe**.
- Préfixe : commence par gl suivie d'une lettre majuscule pour chaque mots définissant la commande.
- Suffixe : détermine le **nombre** et le **type** des arguments.
- Quand il existe plusieurs versions d'une même fonction prenant différents types d'arguments, ces différents fonctions différent par leur suffixe.
- Constantes : Les constantes sont données en majuscule et commencent par le préfixe *GL_*. Exemple :
  - GL_COLOR_BUFFER_BIT

# Exemple

```
/* Couleur d'effacement */
glClearColor();

/* efface l'ecran a l'aide de la couleur
        glClearColor() */
glClear();

/* suffixe 3 : indique qu'il y a 3 parametres
   suffixe f : indique que les parametres
                sont flottants*/
glColor3f(1.0, 1.0, 1.0);

/* Deux parametres entiers */
glVertex2i(1, 3);

/* suffixe v : parametre dans un tableau
   equivalent a glColor3i(1, 0, 1)*/
int c[3]={1, 0, 1}
glColor3iv(c);
```

# Syntaxe des commandes OpenGL

Les types usuels du C sont redéfinis dans OpenGL

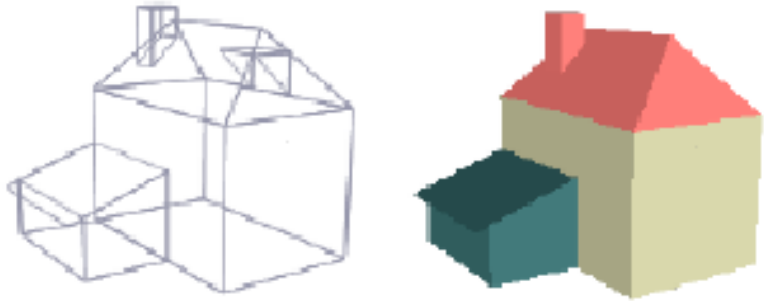
| Suffixe | Type de données | Type C | Type OpenGL |
|---|---|---|---|
| b | Entier 8 bits | signed char | GLbyte |
| s | Entier 16 bits | short | GLshort |
| i | Entier 32 bits | long, int | Glint |
| f | Réel 32 bits | float | GLfloat |
| d | Réel 64 bits | double | GLdouble |
| ub | Entier non signé 8 bits | unsigned char | Glubyte ou GLboolean |
| us | Entier non signé 16 bits | unsigned short | GLushort |
| ui | Entier non signé 32 bits | unsigned long | Gluint ou Glenum |

# Variables d'états

OpenGL est une machine à états

## Exemple

- En modifiant les varaibles d'états, on peut changer le style de rendu :



- Des instructions existent pour placer le système dans certains états. Ceux-ci resteront actifs (utilisés à cette valeur) jusqu'à être changés.
- Chaque fois que le dessin d'un objet est demandé, l'ensemble de ces variables définit la manière avec laquelle l'objet est dessiné.
- L'ensemble de ces "variables d'états" constitue l'environnement OpenGL.
- De nombreuses variables d'états contrôlent l'éclairage, le placage de texture, le style de rendu des primitives, etc.
- Ces variables peuvent être manipulées par les fonctions :
  - glColor*, glNormal*, glTexCoord*
  - glPointSize, glPolygonMode, glShadeModel
  - glEnable, glDisable
  - etc.

# Fonctions de manipulation directe des variables d'état booléenne

- glEnable(GLenum pname) : Activation d'une variable d'état booléenne
  - pname : variable d'état à activer
- glDisable(GLenum pname) : Inhibition d'une variable d'état booléenne
  - pname : variable d'état à désactiver

## Exemple

```
/* activation de la gestion
   des lumieres et des materiaux*/

glEnable(GL_LIGHTING)
```

# Primitives géométriques

La modélisation d'objets sous OpenGL est obtenue uniquement par la définition d'une **liste de primitives** (facettes, lignes et points). Cette contrainte impose de bien connaître la géométrie de l'objet à construire ou à utiliser des logiciels spécialisés pour produire la liste des sommets, arêtes et faces des objets.

# Description des primitives géométriques

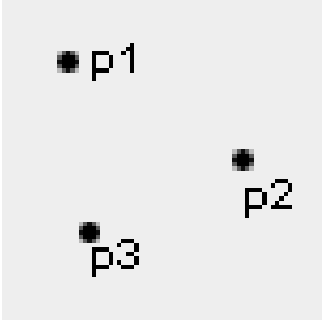
- La description d'objets sous OpenGL repose sur **l'énumération des sommets** formant des primitives géométriques simples.
- Ces primitives simples sont :
  - les points (GL_POINTS),
  - les lignes (GL_LINES)
  - facettes planes à trois cotés (GL_TRIANGLES),
  - quatre cotés (GL_QUADS)
  - n cotés (GL_POLYGON).
- Toutes les primitives géométriques sont décrites en termes de **coordonnées de sommets** (vertex).

- Pour dire à OpenGL de créer un ensemble de points, lignes ou faces on doit placer les sommets entre **glBegin** et **glEnd** :

```
glBegin(mode)
        /* definitions des sommets */
glEnd()
```

- Caractéristiques (Attributs) des sommets :
  - Coordonnées
  - Normale
  - Couleur
  - Coordonnées de texture

# Primitives géométriques : Points




- Les coordonnées sont toujours données en 3D (en 2D, z=0).
- OpenGL travaille en coordonnées homogènes : (x, y, z, w) si w est différent de zéro alors les coordonnées du point dans l'espace sont (x/w, y/w, z/w)
- **Commande** : glVertex{234}{sidf}[v]()
- **Attribut** : glPointSize(GLfloat taille_pixels)

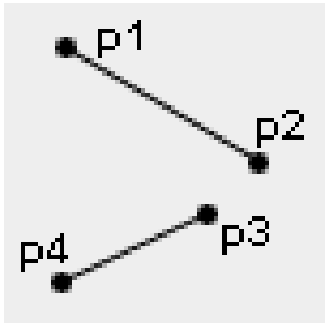
# Exemple

```
glBegin(GL_POINTS) ;
  glVertex2f(-0.5, -0.5) ;
  glVertex2f(-0.5, 0.5) ;
  glVertex2f(0.5, -0.5) ;
  glVertex2f(0.5, 0.5) ;
glEnd ;
```

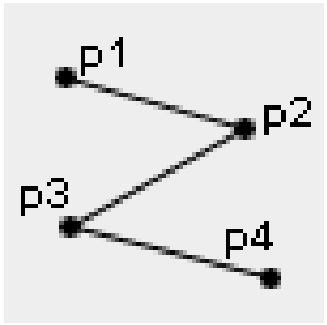
```
Glfloat P1[2]={-0.5, -0.5};
Glfloat P2[2]={-0.5, 0.5};
Glfloat P3[2]={0.5, -0.5};
Glfloat P4[2]={0.5, 0.5};

glBegin(GL_POINTS) ;
  glVertex2fv(P1) ;
  glVertex2fv(P2) ;
  glVertex2fv(P3) ;
  glVertex2fv(P4) ;
glEnd ;
```

**Remarque** : La notation vectorielle (suffixe v) est généralement plus rapide.

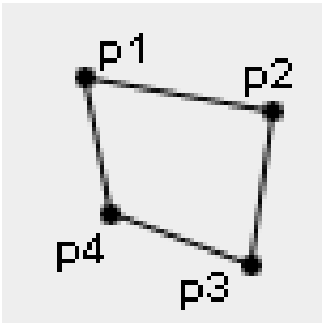
# Primitives géométriques : Lines




GL_LINES




GL_LINE_STRIP

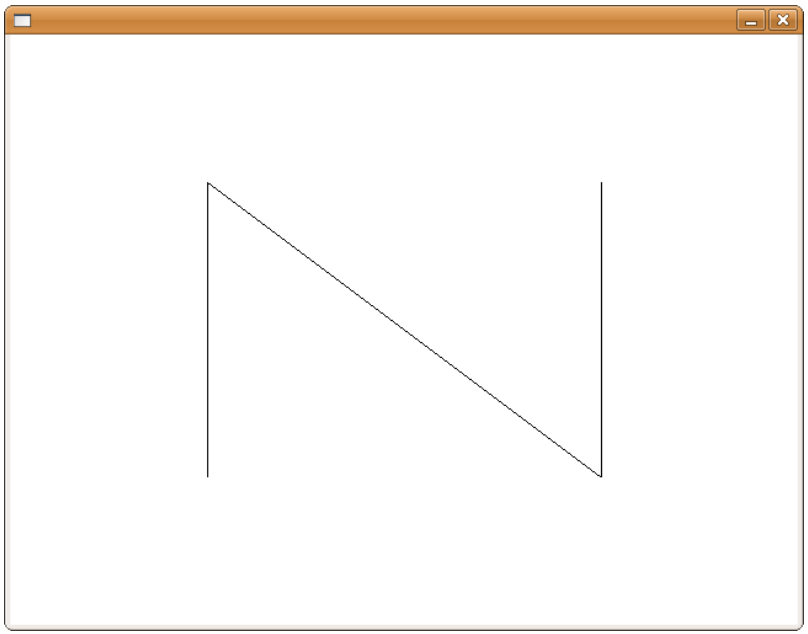



GL_LINE_LOOP

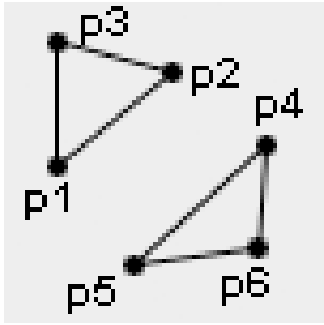
- Il s'agit de segment de droite qui peuvent être connectés.
- Les coordonnées des extrémités sont des sommets.
- **Mode** = GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP
- **Attributs** :
  - glLineWidth(GLfloat taille_pixels)
  - Le tracé de ligne peut être affecté par le mode antialiasing
  - Lignes en pointillé : glLineStipple() (activé avec glEnable(GL_LINE_STIPPLE))

# Exemple

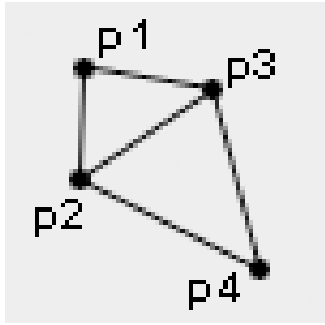
```
glBegin(GL_LINE_STRIP) ;
  glVertex2f(-0.5, -0.5) ;
  glVertex2f(-0.5, 0.5) ;
  glVertex2f(0.5, -0.5) ;
  glVertex2f(0.5, 0.5) ;
glEnd ;
```



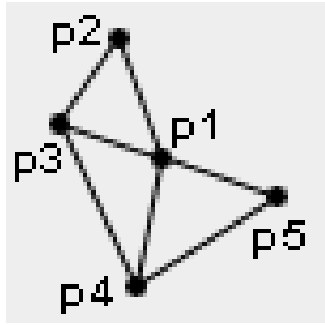
# Autres primitives géométriques : Polygones & cie




GL_TRIANGLES

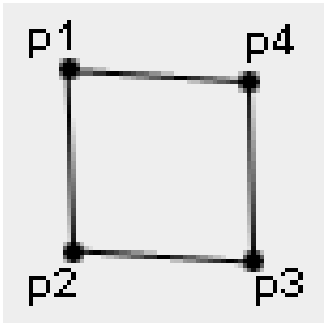



GL_TRIANGLE_STRIP

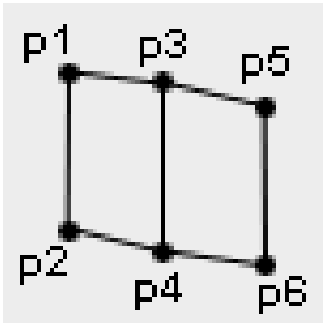



GL_TRIANGLE_FAN

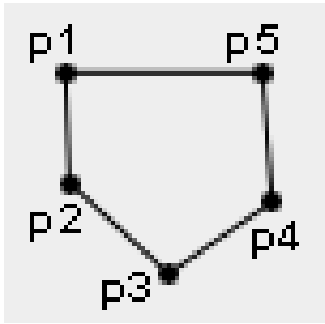
- GL_TRIANGLE_STRIP : Faces triangulaires partageant la dernière arête,
- GL_TRIANGLE_FAN : Faces triangulaires partageant le premier sommet




GL_QUADS




GL_QUAD_STRIP

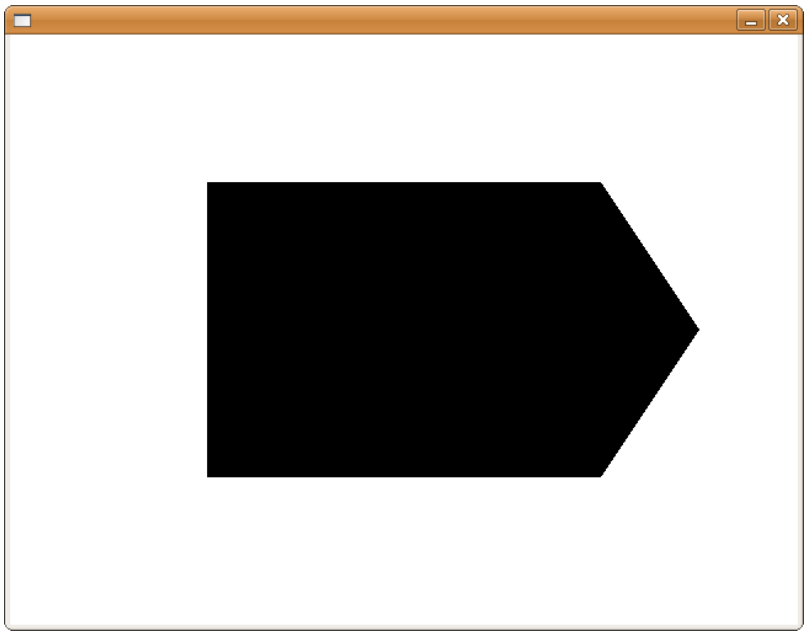



GL_POLYGON

- GL_POLYGON
  - Boucle fermée de segments de droite
  - Uniquement convexes
  - Pas de trous
- GL_QUAD_STRIP : Faces quadrilatères placées sur une grille de sommets)

# Exemple : un polygone

```
glBegin(GL_POLYGON) ;
  glVertex2f(-0.5, -0.5) ;
  glVertex2f(-0.5, 0.5) ;
  glVertex2f(0.5, 0.5) ;
  glVertex2f(0.75, 0.0) ;
  glVertex2f(0.5, -0.5) ;
glEnd();
```



# Gestion de la couleur

- RGB : chaque couleur est codée par **4 composantes** qui définissent l'intensité en R(rouge), G(vert), B(bleue), A(alpha).
- Chaque composante doit varier dans **[0.0, 1.0]**. Cette représentation flottante est privilégiée au niveau hardware, notamment pour les calculs de rendu.
- Une quatrième valeur, dite composante **alpha**, peut être spécifiée. Il s'agit d'un **coefficient d'opacité** qui vaut 1 par défaut. Une valeur plus faible permettra de définir une certaine transparence pour une face et de « voir » les objets qui se trouvent derrière.
- La couleur est considérée comme un état.

## Définition de la couleur d'une face

- La couleur des primitives : *glColor3f(rouge, vert, bleu)*

## La couleur du fond

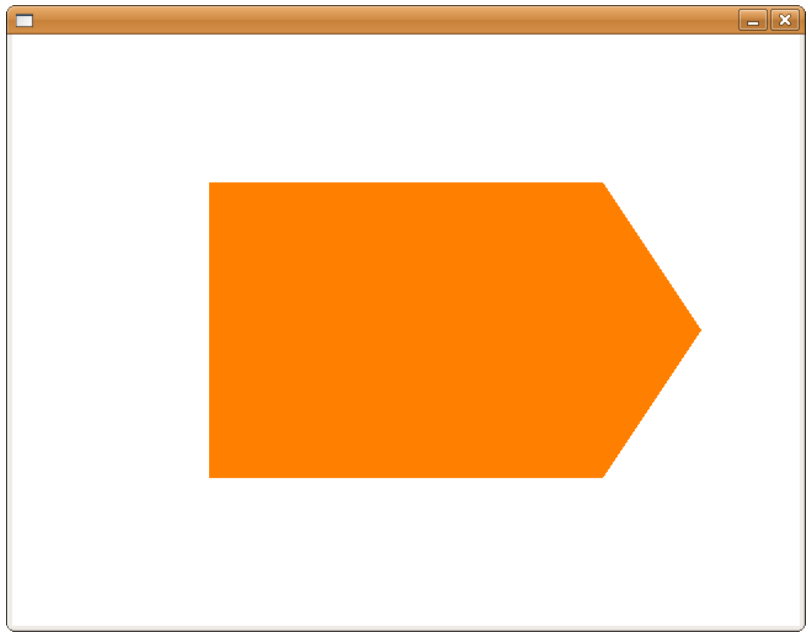
- Définit la couleur d'effacement, ici noir : *glClearColor(0.0,0.0,0.0,1.0)* ;

## Effacement du contenu de la fenêtre

- La couleur du fond est affectée à une variable d'état et sera utilisée pour chaque appel de *glClear(GL_COLOR_BUFFER_BIT)* ;

# Exemple

```
//Couleur de fond en blanc
glClearColor(1.0, 1.0, 1.0, 1.0) ;

//Efface la fenetre, et redessine le fond en blanc
glClear(GL_COLOR_BUFFER_BIT);

//Definit la couleur des primitive en orange
glColor3f(1.0,0.5,0.0);


glBegin(GL_POLYGON) ;
        glVertex2f(-0.5, -0.5) ;
        glVertex2f(-0.5, 0.5) ;
        glVertex2f(0.5, 0.5) ;
        glVertex2f(0.75, 0.0) ;
        glVertex2f(0.5, -0.5) ;
glEnd();
```



# A vous de jouer...