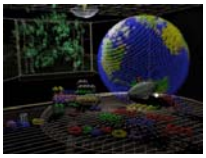
Interfaces de programmation 3D

Gilles Debunne
2004




Objectifs du cours

- Présenter les différentes API existantes
- Expliquer le principe d'utilisation
- Décrire les fonctionnalités les plus courantes

- Interfaces utilisateur (GUI)
- Fournir du code de départ, des exemples

- Fonctionnalitées avancées au cours suivant
- Mise en œuvre par un TP applicatif

Interface de programmation 3D

- Permet d'afficher l'image d'une scène 3D
- Interface avec un langage (C++, Java, Ada…)

- Différentes représentations de la scène
  Primitives de plus ou moins haut niveau
  "3 chaises alignées" … "un pixel bleu en (123,456)"
- Plus ou moins proche du hardware

OpenGL 1.5 ou 2.0

- Bibliothèque bas niveau
- Développée sur *toutes* les architectures
- Intégration avec X11 sur réseau
- Proche du hard, efficace
- Ajouts : GLU, GLUT, GLUI, OpenInventor
- Conception ancienne (ordres, pas d'objet)
- Devra accompagner les évolutions récentes

`www.opengl.org`

DirectX 9.0

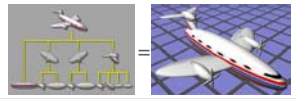
- Bibliothèque objet bas niveau
- Windows seulement (jeux vidéos)
- Repère main gauche
- Usine à gaz d'objets Microsoft
  Flexible Vertex Format, Device (Carte), …
- Intègre les dernières évolutions des cartes

`www.directx.org`

Java 3D 1.3

- Bas et moyen niveau
- Fonctionne en théorie partout
- Graphe de scène complet (tracker, VR, …)
- Bibliothèques (formats, images, interface…)
- Nombreuses classes de base (matr., quat…)
- Compilation de la scène, gestion du son
- Avantages et inconvénients liés à Java

`www.j3d.org`



Performer

- Bibliothèque moyen niveau professionnelle
- Stations SGI et Linux
- Gère le parallelisme (plusieurs pipelines)
- Gestion de scènes complexes
- Charge de nombreux formats de fichiers 3D
- Graphe de scène
- Crée et gère ses fenêtres
- Plus vraiment maintenu, mal documenté

`www.sgi.com/software/performer/`

Open Inventor

- Surcouche moyen niveau de OpenGL
- SGI, portage windows payant
- Graphe de scène
- Visualiseur avec sélection, extensible
- Liée au format `wrl` (VRML 1 & 2)
- Pas très efficace

Quelle API choisir ?

- Visualiser un objet 3D : difficile (cf formats)
  Maya, 3D StudioMax,
  `perfly` (Performer), `ivview` (Inventor), Java3D
- Petite application graphique non critique
  OpenGL et libQGLViewer
  Java3D (bibliothèque de classes réutilisables)
- Application graphique performante
  OpenGL, DirectX

Plan du cours

- Présentation des différentes API 3D
- Structure d'une application graphique
  Illustration en OpenGL, DirectX est assez proche
  Création d'un contexte OpenGL
  Placement des repères
  Affichage de primitives
- Notions avancées

Notation des fonctions

- `glCeciCela(…);`
- Constantes : `GL_NOM_DE_CONSTANTE`
- Nombre d'arguments : 2, 3 ou 4
- Type : `f` float, `d` double, `b` byte, `i` int, `l` long,…

```
glVertex2i(1, 3);
glVertex3f(1.62, 3.34, 5.17);
glColor4f(0.2, 0.4, 0.6, 0.5);
glRotatef(…);
```

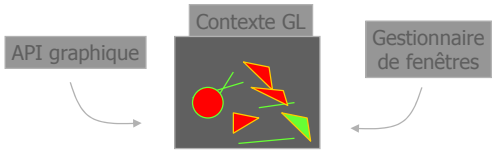
- **`v`** : par adresse

```
double coord[3];
glVertex3dv(coord);
```


Structure d'un programme

- Demande de création d'un contexte GL
  Double buffer, stéréo, transparence, …
- Procédure principale en boucle
  Gestion de la souris et du clavier
  Procédure d'affichage
    Effacer l'écran
    Description de la caméra
    Placer et dessiner chaque objet

Le contexte GL



- Machine à états
  - Matrices de transformation, type d'affichage, couleurs, normale, …
- Buffers (color, depth, stencil, …)
  - Stockant, pour chaque pixel, plus ou moins de bits

Création d'une fenêtre openGL

- GLUT (GL Utility Toolkit)

```
int glutCreateWindow(char* name);
glutDisplayFunc(display);
glutMainLoop();
```

- Qt (www.trolltech.com)

  Dériver de la classe `QGLWidget` et surcharger les méthodes `paintGL(),keyPressEvent(),…`

  Utiliser `libQGLViewer`

Procédure d'affichage principale

- Effacer l'écran
- Description de la caméra
- Pour chaque objet de la scène
  - Placement de l'objet
  - Modification de la machine à état
  - Ordres d'affichage

Effacer l'écran

- Couleur de fond

```
glClearColor(r,g,b,a);
```

- Effacement

```
glClear(flags);
```

flags = { `GL_COLOR_BUFFER_BIT` | `GL_DEPTH_BUFFER_BIT` | `GL_ACCUM_BUFFER_BIT` | `GL_STENCIL_BUFFER_BIT` }

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```


Description de la caméra

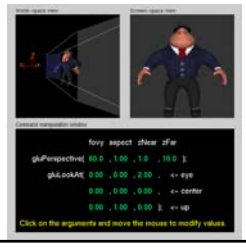
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
```

`glOrtho(…);` ou `glFrustum(…);`

Ou `gluPerspective();` et `gluLookAt(…);`

Matrice 4x4

Passage de 3D à 2D½



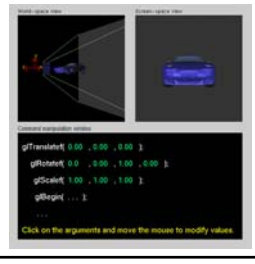
Positionnement des objets

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(x,y,z);
glRotatef(alpha, x,y,z);
glScalef(sx, sy, sz);
```

OU `glMultMatrixf(m);`

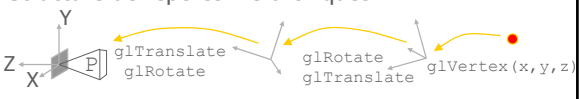
```
glBegin(…);
…
glEnd();
```

Ordre inversé !



Pile de matrices

- Structure de repères hiérarchiques



$P(x,y,z) = P * T_1 * R_1 * R_2 * T_2 * (x,y,z)$

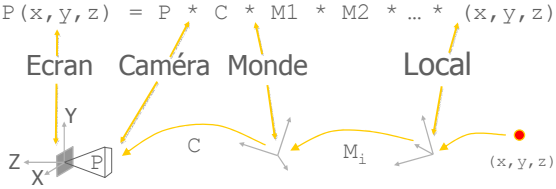
- Pour chaque `glMatrixMode`, on a une pile

```
glPushMatrix();
// Modification et utilisation de la matrice
glPopMatrix();
```

Très utile pour le dessin de structures hiérarchiques

libQGLViewer

`P(x,y,z) = P * C * M1 * M2 * … * (x,y,z)`

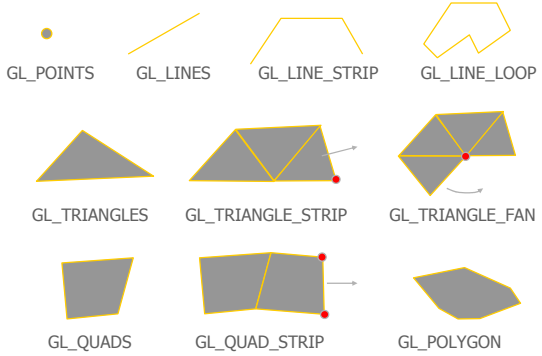


`P` géré via les paramètres de `camera()`

`C` modifiée par la souris

Dans le repère monde au début de `draw()`

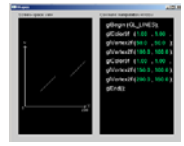
Primitives graphiques : `glBegin(…)`



Affichage de primitives

```
glBegin(primitive);
  // primitive = GL_POINTS, GL_LINES, GL_TRIANGLES…
  glColor3f(r,g,b);
  glNormal3f(nx, ny, nz);
  glTexCoord(u,v);
  glVertex3f(x,y,z);
  …
glEnd();
```

} Répété *n* fois (*n* selon la *primitive*)



Seul le `glVertex` est obligatoire,
sinon la dernière valeur spéficiée est utilisée.

Plan du cours

- Présentation des différentes API 3D
- Structure d'une application graphique
- Notions avancées
  - Machine à état
  - Matériaux, lumières, transparence
  - Brouillard
  - Display List, lecture du frame buffer
  - Sélection

OpenGL : machine à états

- Valeurs scalaires

```
glClearColor(r,g,b,a);
glMatrixMode(GL_MODELVIEW);
glMultMatrixd(mat);
glNormal3f(nx,ny,nz);
glColor3f(r,g,b);
…
```

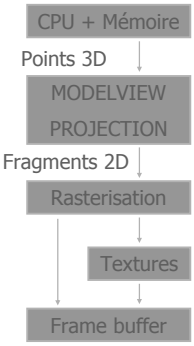
- Valeurs booléennes

```
glEnable(flag);    glDisable(flag);
flag = GL_DEPTH_TEST, GL_COLOR_MATERIAL, GL_LIGHTING,
       GL_BLEND, GL_CULL_FACE, GL_TEXTURE_2D, …
```

- Sauvegarde, restauration, lecture

```
glPushAttrib(flag);    glPopAttrib();
glIsEnabled(…);  glGetFloatv(…); …
```

Le pipeline graphique



Rasterisation

Z-buffer : élimination des parties cachées

Clipping hors frustum

Interpolation des informations aux sommets

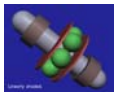
La machine à état correspond aux réglages des étages du pipeline

Options d'affichage

```
glLineWidth(3);
glPointSize(12);
```

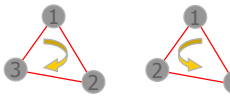
```
glEnable(GL_LIGHTING);
```

Prise en compte de la normale et des lumières



```
glPolygonMode(face, mode);
mode = GL_FILL, GL_LINE
```

```
glCullFace(face);
glEnable(GL_CULL_FACE);
face = GL_FRONT, GL_BACK
```



Matériaux des objets

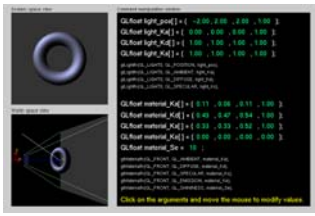
Couleur ambiente, diffuse, spéculaire, émission

Pour les faces avant et/ou arrière

```
glMaterialfv(GL_FRONT, GL_SPECULAR, spec);
```

Multiplication composante par composante : non physique

Voir la doc de `glLightModel()`



glColorMaterial

face = GL_FRONT, GL_BACK, **GL_FRONT_AND_BACK**

material = GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION, **GL_AMBIENT_AND_DIFFUSE**

```
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(face, material);
glColor3f(r,g,b);
```

- `glColor` court-circuite `GL_AMBIENT` et `GL_DIFFUSE` lorsque `GL_LIGHTING` est activé
- Et est utilisé lorsque `GL_LIGHTING` est désactivé !

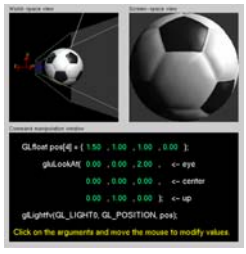
Lumières

- Sources à l'infini, directionelles, spots

- Normales du modèle nécessaires

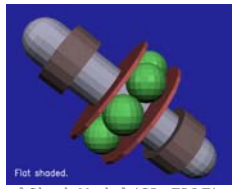
```
glLightfv(GL_LIGHT0, GL_POSITION, pos);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

- Objet comme les autres
  → subit la `MODELVIEW`



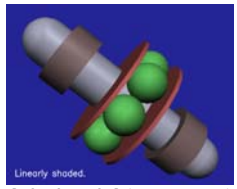
Interpolation des couleurs

Une couleur par élément
(la dernière spécifiée)



`glShadeModel(GL_FLAT);`

Interpolation linéaire
(sur RGB) des valeurs aux sommets



`glShadeModel(GL_SMOOTH);`

Raisonnable pour l'éclairage diffus, problématique pour le spéculaire :
→ Interpolation de la normale
→ Suréchantillonage

Transparence α=opacité

- Combinaison des couleurs
  - Fragment dessiné $c_1=(r_1,g_1,b_1,\alpha_1)$
  - Couleur actuelle du pixel $c_2=(r_2,g_2,b_2,\alpha_2)$

$$c = \alpha_1 c_1 + (1-\alpha_1) c_2$$

- Différentes fonctions de mélange

```
glBlendFunc(source, dest);
glEnable(GL_BLEND);
```

- Buffer RGBA non nécessaire (en back to front)

Transparence : difficultés

- Limitations
  - Approximation de la physique
  - Rendu de l'arrière vers l'avant
    - Sauf si une seule épaisseur est transparente
- Mauvaise fonction de mélange par défaut

  Utiliser `glBlendFunc(`α_{src}`, 1-`α_{src}`);`
  ou mieux (si toutes les couleurs sont *premultiplied alpha*)
  `glBlendFunc(1, 1);`

Les buffers

- Frame-buffer : couleur (r,g,b,[a])
  - Simple ou double, stéréo

```
glDrawBuffer(GL_FRONT ou GL_BACK, …);
```

- Profondeur (Z-buffer)
- Pochoir (Stencil buffer)
- Accumulation

- Sous-fenêtrage

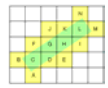
```
glScissor(…);   glViewport(…);
```


Anti-aliassage

- Utilisation du canal alpha

```
glEnable(GL_{POINT, LINE, POLYGON}_SMOOTH);
```

- Suréchantillonnage
  - Rendu d'une fenêtre plus grande
  - Anti-aliassage en hardware



- Nvidia sur Linux

```
setenv __FSAA_MODE n   (n ∈ 1..5)
glEnable(GL_MULTISAMPLE_ARB);
```


Brouillard

- Limite le nombre de primitives à afficher
- Accentue l'effet de profondeur

Décalage d'affichage

- Clignotement à l'affichage
  - Deux primitives superposées (surface & fil de fer)
  - Limite de précision du z-buffer

```
glEnable(GL_POLYGON_OFFSET_{LINE,POINT,FILL});
glPolygonOffset(facteur, unités);
```

- Valeurs positives (par défaut) augmentent le Z
  - La primitive est dessinée derrière

Display Lists

- Optimisation de l'affichage
  Plus de transfert, données stockées sur la carte

```
// Création de la display list
Gluint objetDL = glGenList(1);

glNewList(objetDL, GL_COMPILE);
  // Ordres GL
glEndList();

// Rappel de la display list, affichage
glCallList(objetDL);
```

- Géometrie identique à chaque fois
- Extraire les attributs spécifiques
  Mode d'affichage, matrice de `MODELVIEW`, couleur...

Récupération de l'image

- Lecture pixel par pixel
  `glReadPixel(…)`
- Rendu hors écran
  Offscreen rendering, Pbuffer
- Copie directe dans une texture

```
glBindTexture(GL_TEXTURE_2D, tex);
glCopyTexSubImage2D(GL_TEXTURE_2D,
        0,        // Mipmap level
        0, 0,     // Offset
        0, 0,     // Origin
        tex, size);
```

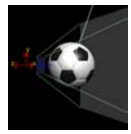

Sélection

Voir exemple `select`

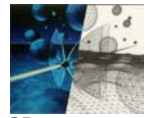
- Savoir quel est l'objet sous la souris
- Rendre la scène (pas d'affichage)
  `glRenderMode(`*mode*`)` `gluPickMatrix(…)`
  Limitation à une région de quelques pixels
  Etiquetage des primitives :
  `glInitNames(); glPushName(); glPopName();`
- Interpréter les résultats
  Pile d'étiquettes
  Autres informations (clipping, couleur, z, …)

Plan de clipping additionnels

- Matrice de projection = 6 plans de clipping



- 6 plans additionnels possibles
  Rendu plus lent
  Calculs complexes sinon
  Permettent la vue de coupes d'objets 3D

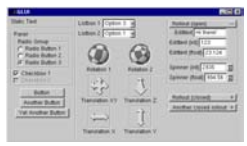


Plan du cours

- Présentation des différentes API 3D
  OpenGL, DirectX, Java3D, Performer, Inventor
- Structure d'une application graphique
- Notions avancées
- Interface utilisateur (GUI)
  GLUI
  Qt
  Windows
  Java

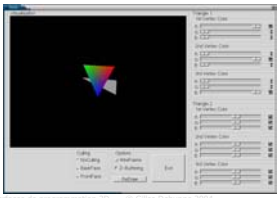
GLUI

- Simple mais pas très objet
  `(void*)`
  Variables globales
- Windows et Unix-Linux



Qt 3.3

- Multi-plateformes (Unix, Windows, Mac, …)
- `qmake` (Makefile), `designer` (UI à la souris)
- Passage de messages entre objets :
  Impossible en C++ → précompilateur MOC

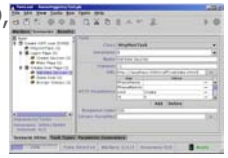


Windows

- Interface integrée Windows
  Component Object Modeling
  Communication par messages
  Menus, cases à cocher, souris, clavier…
- Deux systèmes possibles
  Microsoft Fondation Classes
  Win32

Java

- Classes d'interface utilisateur
  Indépendant de Java 3D
  Bibliothèque JFC Swing (remplace AWT)
- Java3D intègre les classes liées à la RV
  Tracker, lunettes 3D, souris 3D, gants, matrices, quaternions…
  Monde physique-monde virtuel



"The JFC Swing Tutorial" K. Walrath, M. Campione

Pointeurs

- Ressources locales
  `/var/shareusers/redbook.pdf` et
  `/var/shareusers/TPs` (sur `mbosun`)
  `/usr/local/share/doc/QGLViewer/examples` (sur `mboarv`)
- Code à copier-coller
  `artis.imag.fr/~Gilles.Debunne/Enseignement/3DAPI`
- Page web du cours (autres cours, sujets de TPs)
  `artis.imag.fr/~Gilles.Debunne/Enseignement`

Exemple de programme

```
#include "simpleViewer.h"
#include <qapplication.h>

int main(int argc, char** argv)
{
  // Read command lines arguments.
  QApplication application(argc,argv);

  // Instantiate the viewer.
  Viewer v;

  // Make the viewer window visible on screen.
  v.show();

  // Set the viewer as the application main widget.
  application.setMainWidget(&v);

  // Run main loop.
  return application.exec();
}
```


simpleViewer.h

```
#include <QGLViewer/qglviewer.h>

class Viewer : public QGLViewer
{
protected :
  virtual void draw();
  virtual void init();
  virtual QString helpString() const;
};
```

simpleViewer.cpp (1/2)

```
#include "simpleViewer.h"
#include <math.h>

void Viewer::init()
{
  // Restore previous viewer state.
  restoreFromFile();

  // Open help window
  help();
}

QString Viewer::helpString() const
{
  QString text("<h2>S i m p l e V i e w e r</h2>");
  text += "Description de l'exemple en <b>html</b>. ";
  return text;
}
```


simpleViewer.cpp (2/2)

```
void Viewer::draw()
{
  const float nbSteps = 500.0;

  glBegin(GL_QUAD_STRIP);
  for (float i=0; i<nbSteps; ++i)
    {
      float ratio = i/nbSteps;
      float angle = 21.0*ratio;
      float c = cos(angle);
      float s = sin(angle);
      float r1 = 1.0 - 0.8*ratio;
      float r2 = 0.8 - 0.8*ratio;
      float alt = ratio - 0.5;
      const float nor = .5;
      const float up = sqrt(1.0-nor*nor);
      glColor3f(1.0-ratio, 0.2 , ratio);
      glNormal3f(nor*c, up, nor*s);
      glVertex3f(r1*c, alt, r1*s);
      glVertex3f(r2*c, alt+0.05, r2*s);
    }
  glEnd();
}
```


Résultat