Costruzione di Interfacce
Lezione 18
MFC e OpenGL

cignoni@iei.pi.cnr.it
http://vcg.iei.pi.cnr.it/~cignoni

OpenGL e MFC

- In windows tutte le operazioni di disegno sono riferite a Device Context
- Per disegnare usando OpenGL abbiamo bisogno anche di un **OpenGL Rendering Context**
  - È legato a quel device context
  - Deve essere adatto al Device Contex della finestra su cui vogliamo disegnare.

Device Context e Rendering Context

- Tutti i comandi opengl passano attraverso un rendering context (specificato implicitamente)
- Tutti i comandi di grafica in Windows GDI passano attraverso un device context (specificato esplicitamente)
- Un rendering context è legato ad un device context e ne condivide lo stesso pixel format (anche se non è detto che sia lo stesso di quando lo abbiamo creato)
- Un thread può avere un solo rendering context corrente e un rendering context può essere attivo per un solo thread.

Pixel Format

- Ogni device context in windows ha un suo pixel format intrinseco.
- Può essere cambiato una sola volta per finestra
- In un pixelformat l'utente specifica:
  - Numero bit colore
  - Bit z-buffer
  - Se la finestra è double buffered
  - Se c'è uno stencil buffer
  - Ecc.

Pixel Format

- Notare che il pixel format è:
  - Il rendering context che si crea partendo da un dc ne condivide il pixelformat
  - Non tutti i pixelformat sono possibili
  - Quello che si può ottenere è abbastanza dipendente dalle caratteristiche del device context (e dalla modalità video corrente)
  - Non tutti i pixel format sono ammissibili
    - ChoosePixelFormat
    - SetPixelFormat
  - Si sceglie un pixel format dando un indice.
  - Non tutti I pixel format sono accelerati hw

PixelFormat

Step tipici:

- Riempire una struttura *pixelformat descriptor*

```
PIXELFORMATDESCRIPTOR pfd = {
     sizeof(PIXELFORMATDESCRIPTOR), // size of this pfd
     1, // version number
     PFD_DRAW_TO_WINDOW | // support window
     PFD_SUPPORT_OPENGL | // support OpenGL
  …
};
```

- Chiedere un pixelformat che assomigli a quello chiesto:

```
int iPixelFormat = ChoosePixelFormat(hdc, &pfd);
```

- (opz) controllare cosa è ritornato
- Settare il pixelformat ottenuto

```
SetPixelFormat(hdc, iPixelFormat, &pfd);
```

Esempio PixelFormat

```
PIXELFORMATDESCRIPTOR pfd = {
sizeof(PIXELFORMATDESCRIPTOR),  //  size of this pfd
     1,                     // version number
PFD_DRAW_TO_WINDOW |   // support window
PFD_SUPPORT_OPENGL |   // support OpenGL
PFD_DOUBLEBUFFER,      // double buffered
PFD_TYPE_RGBA,         // RGBA type
24,                    // 24-bit color depth
0, 0, 0, 0, 0, 0,      // color bits ignored
0,                     // no alpha buffer
0,                     // shift bit ignored
0,                     // no accumulation buffer
0, 0, 0, 0,            // accum bits ignored
32,                    // 32-bit z-buffer
0,                     // no stencil buffer
0,                     // no auxiliary buffer
PFD_MAIN_PLANE,        // main layer
0,                     // reserved
0, 0, 0                // layer masks ignored
};
```

Rendering Context

- Una volta che abbiamo una finestra con il pixelformat che ci è appropriato si può creare il rendering context opengl
- HGLRC hglrc=wglCreateContext(hdc);
- Per settare il rendering context corrente del thread
- wglMakeCurrent(hdc, hglrc);

Stili e classe finestre e opengl

- Quando in Windows si crea una finestra se ne stabilisce lo stile
- Per aver Opengl in una finestra occorre che questa abbia come stile:
- WS_CLIPCHILDREN
  - Quel che si disegna nella finestra padre non influenza i figli
- WS_CLIPSIBLINGS
  - Idem per i fratelli
- E non avere
- CS_PARENTDC
  - altrimenti la finestra userebbe il device context del padre

OpenGL e MFC

- Dove va tutto cio?
- In un'applicazione con architettura doc/view, si aggiunge quel che serve nella classe view.
  - Alla creazione della finestra ci assicuriamo che la finestra abbia lo stile giusto
  - Prendiamo un device context per la client area della finestra
  - Scegliamo e settiamo il pixel format
  - Creiamo e ci salviamo un handle ad un rendering context opengl

Modifica della classe

- Include nel .h della classe view
- `#include <GL/gl.h>`
- `#include <GL/glu.h>`
- Opengl rendering context e device context
- `CClientDC *m_pDC; // DC della finestra`
- `HGLRC m_hrc;  // Contesto OpenGL`

Stile

- Si può cambiare con cui viene create una finestra MFC modificando la PreCreateWindow

```
BOOL MyOpenGLView::PreCreateWindow(CREATESTRUCT& cs)
{
   // An OpenGL window must be created with the following flags and must not
   // include CS_PARENTDC for the class style. Refer to SetPixelFormat
   // documentation in the "Comments" section for further information.
   cs.style |= WS_CLIPSIBLINGS | WS_CLIPCHILDREN | CS_OWNDC;
   if(cs.style&CS_PARENTDC)
   {
        AfxMessageBox("Win OpenGL Internal Error!");
        return FALSE;
   }

   return CView::PreCreateWindow(cs);
}
```

Pixelformat e rendering context

- Va fatto subito dopo la creazione della finestra. Override (esplicito) della Create()

```
BOOL MyOpenGLView::Create(LPCTSTR lpszClassName,
        LPCTSTR lpszWindowName, DWORD dwStyle, const RECT& rect,
        CWnd* pParentWnd, UINT nID, CCreateContext* pContext)
{
  if( ! CView::Create(lpszClassName, lpszWindowName, dwStyle,
                rect, pParentWnd, nID, pContext))
        return false;

  m_pDC = new CClientDC(this);
  ASSERT(m_pDC != NULL);

  // Setta il pixel format
  if (!SetupPixelFormat(m_pDC)) return FALSE;
  // Crea e setta il contesto OPENGL
  m_hrc = wglCreateContext(m_pDC->GetSafeHdc());
  if(m_hrc==NULL){
        AfxMessageBox("OpenGL contest fail");
        return FALSE;
   }
```

Pixel Format

```
BOOL SetupPixelFormat( CDC * pDC )
{
    static PIXELFORMATDESCRIPTOR pfd =
    {
     … omissis …
    };
    int pixelformat;

    if ( (pixelformat = ChoosePixelFormat(pDC->GetSafeHdc(), &pfd)) == 0 )
    {
        AfxMessageBox("ChoosePixelFormat failed");
       return FALSE;
    }

    if (SetPixelFormat(pDC->GetSafeHdc(), pixelformat, &pfd) == FALSE)
    {
        AfxMessageBox("SetPixelFormat failed");
       return FALSE;
    }

    return TRUE;
}
```

Librerie

- Fatto questo se si compila il linker non risolve alcuni simboli
- Aggiungere le librerie
  - Property page->linker->input -> Additional dependencies
  - Opengl32.lib
  - Glu32.lib

OnDestroy

- Se non si fa pulizia all'uscita si hanno memory leaks…
- Gestiamo il messaggio WM_DESTROY mandato alla finestra quando questa viene distrutta

```
void myview::OnDestroy()
{
   CView::OnDestroy();
   wglMakeCurrent(m_pDC->GetSafeHdc(),m_hrc);
   glFinish();
   wglMakeCurrent(NULL,  NULL);
  if (m_hrc) ::wglDeleteContext(m_hrc);
  if (m_pDC) delete m_pDC;
}
```

OnEraseBackGround

- Il clear del background non deve piu'essere fatto automaticamente, ma direttamente con una funzione OpenGL.
- Si deve intercettare la pulizia automatica del background di una finestra
- Handler messaggio WM_ERASEBKGND

```
BOOL myview::OnEraseBkgnd(CDC* pDC)
{
   return true; // lo facciamo noi…
}
```

Finalmente OnDraw

- Adesso possiamo disegnare con opengl nella onDraw!!

```
void CMoebius3View::OnDraw(CDC*  pDC)
{
   wglMakeCurrent(m_pDC->GetSafeHdc(),m_hrc);
   glClear(GL_COLOR_BUFFER_BIT);
   GLfloat triangle[3][2]={{-1.0f,-1.0f},{1.0f,-1.0f},{ 0.0f, 1.0f}};
   GLfloat p[2]={0.0, 0.0f};
   int i, j;
   glClear (GL_COLOR_BUFFER_BIT);
   glBegin(GL_POINTS);
   for(j=0;j<20000;j++)        {
        glVertex2f(p[0],p[1]);
        i=rand()%3;
        p[0]=(p[0]+triangle[i][0])/2.0f;
        p[1]=(p[1]+triangle[i][1])/2.0f;
   }
   glEnd();

   glFlush();
   SwapBuffers(m_pDC->GetSafeHdc());
}
```

Riassunto

- PreCreateWindow
  - Lo stile giusto
- Create
  - Pixelformat e wglCreateContext
- OnEraseBackgnd
  - Ritornare true
- OnDraw
  - Classico disegno e swapbuffer
- OnDestroy
  - Distruzione contesti vari

OnSize

- Al solito copiamo dal vecchio prog glut

```
void CMoebius3View::OnSize(UINT nType, int cx, int cy)
{
  wglMakeCurrent(m_pDC->GetSafeHdc(),m_hrc);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  float ratio=(float)cx/(float)cy;
  glOrtho(-1,1,-ratio,ratio,-1,1);
  glViewport (0, 0, (GLsizei) cx, (GLsizei) cy);
  glMatrixMode (GL_MODELVIEW);    /*  back to modelview
  matrix    */
}
```

- Non va! perche?

OnSize 2

- OnSize viene chiamata prima che ci sia il contesto opengl!
- Conviene correggere il costruttore (init m_pDC e m_hrc a 0) e incapsulare la wglMakeCurrent in una funzione che faccia qualche test e togliere tutte le wglMakeCurrent e sostituirle con SetGL (testando se fallisce…)

```
inline BOOL SetGL()
{
  if(!(m_pDC && m_hrc)) return false;
  if(!wglMakeCurrent(m_pDC->GetSafeHdc(), m_hrc)){
       AfxMessageBox("GlMakeCurrent Error");
       return FALSE;
    }
  return TRUE;
}
```

OnDraw e thread

- Nelle MFC il ridisegno è fatto in un thread differente.
- Conviene prevedere un meccanismo per evitare di disegnare quando l'app sta modificando il documento
- Si aggiunge
  - nel Doc un campo bool busy
  - Nella on draw, all'inizio
    - if(pDoc->busy) return;

Esercizio

- Portare l'esempio Moebius2 sotto MFC.
- Nel doc ci starà un oggetto della classe mesh e il metodo per costruirlo.
- Alla classe mesh si aggiunge un metodo per disegnarsi
- Nella view la OnDraw accederà a tale metodo.