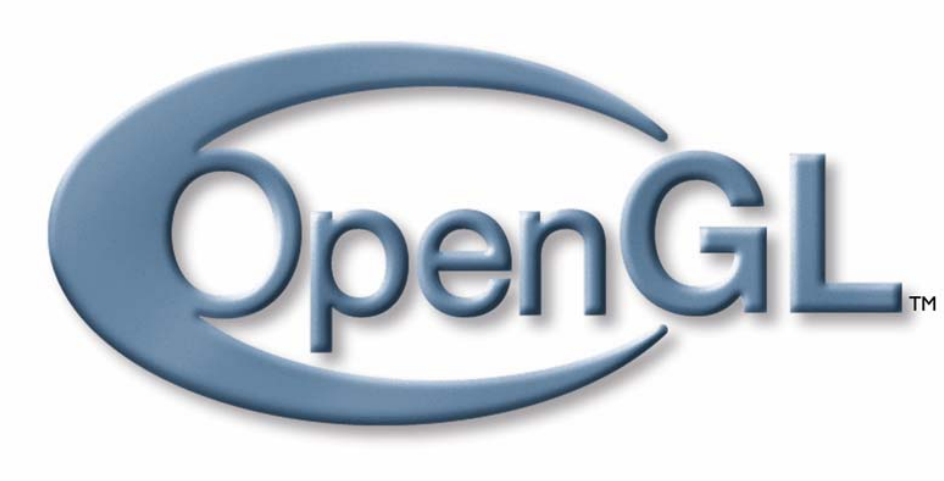
haute école neuchâtel berne jura arc ingénierie saint-imier le locle porrentruy

COURS D'INFOGRAPHIE
Utilisation avec l'API Windows

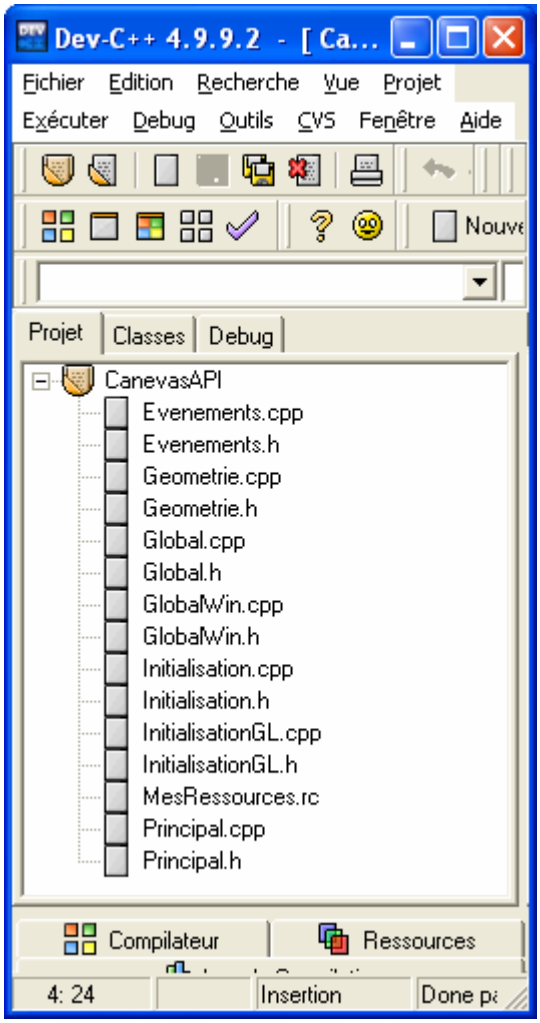


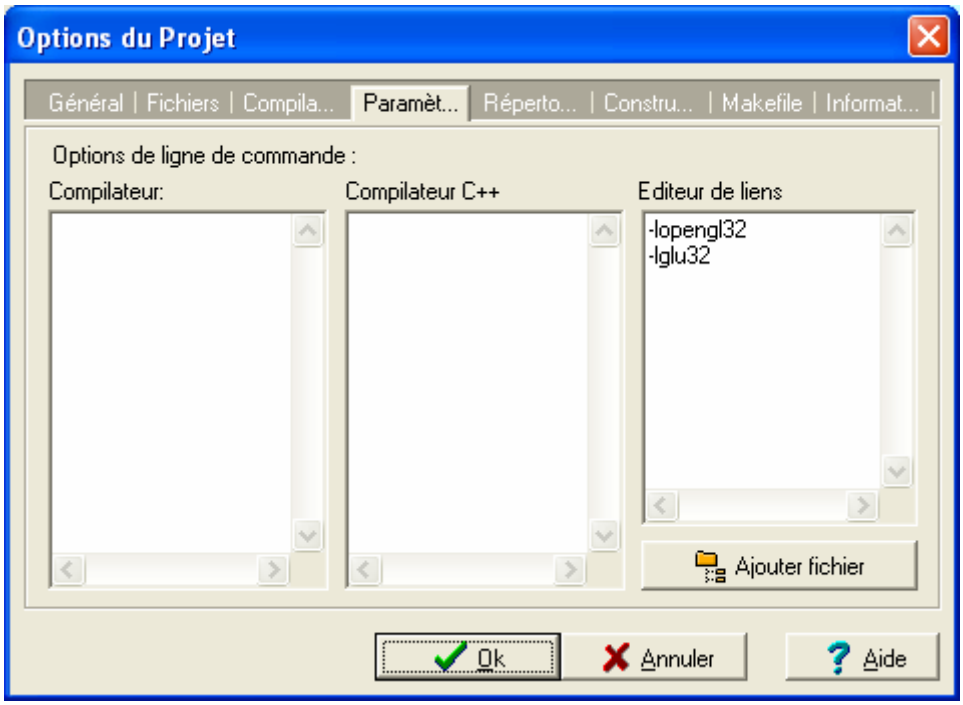
Mai 2006, Pierre Chatelain

Hes·so
Haute Ecole Spécialisée de Suisse occidentale

Canevas d'un programme OpenGL avec API

Les fichiers dans un projet avec DEV-CPP :





Remarques :

Il existe deux groupes de fichiers. Un premier groupe qui contient les instructions qui sont propres à l'API Windows. Un deuxième groupe qui contient les instructions propres à OpenGL. Ce deuxième groupe est en principe portable au niveau des sources.

Fichiers du premier groupe :

- Principal
- Initialisation
- GlobalWin
- Les fichiers de ressources.

Fichiers du deuxième groupe :

- InitialisationGL
- Global
- Evenements
- Geometrie

Attention !

Dans le fichier Evenements, quelques instructions sont propres à l'API Windows. Il s'agit de :

Dans void Display() :

- SwapBuffers(hDC);

Dans void EvKeyDown(unsigned char key) :

- KillGLWindow();
- CreateGLWindow("Canevas OpenGL avec l'API Windows",800,600,16,fullscreen)
- PostQuitMessage(0);
- SetTimer(hWnd, ID_TIMER, 10, NULL)
- KillTimer(hWnd, ID_TIMER)
- SetWindowText(hWnd, Texte)
- MessageBox(hWnd, "SetTimer Failed", "Error", MB_OK)
- InvalidateRect(hWnd, NULL, FALSE)
- Les symboles des touches spéciales (VK_F1, VK_ESCAPE, VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT).

Dans void EvMouseMove(int x, int y, int & xInit, int & yInit,
bool ButtonLeftDown, bool ButtonRightDown)

- InvalidateRect(hWnd, NULL, FALSE);

Dans void Animation(int Id) :

- InvalidateRect(hWnd, NULL, FALSE)

L'initialisation du programme se fait avec le code suivant :

```
// Fichier Principal.cpp

#include <windows.h>      // Header File For Windows
#include <gl\gl.h>        // Header File For The OpenGL32 Library
#include <gl\glu.h>       // Header File For The GLu32 Library
#include "globalwin.h"
#include "global.h"
#include "MesRessources.rh"
#include "evenements.h"
#include "geometrie.h"
#include "initialisation.h"
#include "principal.h"
int xInit;
int yInit;
bool ButtonLeftDown = false;
bool ButtonRightDown = false;

//************************************************************************
// Procédure de fenêtre : WndProc
// Cette fonction est appelée lorsqu'un événement survient.
//************************************************************************
LRESULT CALLBACK WndProc(HWND hWnd,   // Handle For This Window
                UINT uMsg,            // Message For This Window
                WPARAM wParam,        // Additional Message Information
                LPARAM lParam)        // Additional Message Information
{
  switch (uMsg)                       // Check For Windows Messages
  {
    case WM_ACTIVATE:                 // Watch For Window Activate Message
    {
      if (!HIWORD(wParam))            // Check Minimization State
      {
        active=TRUE;                  // Program Is Active
      }
      else
      {
        active=FALSE;                 // Program Is No Longer Active
      }
      return 0;                       // Return To The Message Loop
    }

    case WM_SYSCOMMAND:
    {
      switch (wParam)
      {
        case SC_SCREENSAVE:
        case SC_MONITORPOWER:
       return 0;
      }
      break;
    }

    case WM_COMMAND:
    {
     switch(LOWORD(wParam))
     {
       case ID_FILE_EXIT:
            PostMessage(hWnd, WM_CLOSE, 0, 0);
        return 0;
      }
  }
```

```
     case WM_CREATE :
     {
       if(animationOn)
       {
         // Pour plus de précision, il faudrait utiliser timeSetEvent()
         if(!SetTimer(hWnd, ID_TIMER, 10, NULL))
          MessageBox(hWnd, "SetTimer Failed", "Error", MB_OK);
      }
       return 0;
     }

    // WM_CLOSE est envoyé par le bouton de fermeture et par <ALT><F4>
    // On peut faire des tests ici, p. ex. demander de sauver un fichier.
     case WM_CLOSE:            // Did We Receive A Close Message?
     {
       PostQuitMessage(0);     // Send A Quit Message
       return 0;               // Jump Back
     }
/*
    // Il ne faut pas intercepter le message WM_DESTROY ici
    // car quand on passe en mode fullScreen, la fonction envoie
    // justement un message WM_DESTROY.
     case WM_DESTROY :
     {
       PostQuitMessage(0);           // Send A Quit Message
       return 0;
     }
*/
     case WM_KEYDOWN:                // Is A Key Being Held Down?
     {
       EvKeyDown(wParam);
       return 0;                     // Jump Back
     }

     case WM_KEYUP:                  // Has A Key Been Released?
     {
       return 0;                     // Jump Back
     }

     case WM_SIZE:                   // Resize The OpenGL Window
     {
       ReSizeGLScene(LOWORD(lParam),HIWORD(lParam));  // LoWord=Width,
                                                      // HiWord=Height
       return 0;                     // Jump Back
     }

     case WM_LBUTTONDOWN :
     {
       SetCapture(hWnd);
       ButtonLeftDown = true;
       xInit = LOWORD ( lParam );
       yInit = HIWORD ( lParam );
       return 0;                     // Jump Back
     }

     case WM_LBUTTONUP :
     {
       ReleaseCapture();
       ButtonLeftDown = false;
       return 0;                     // Jump Back
     }
```

```
    case WM_RBUTTONDOWN :
    {
     SetCapture(hWnd);
     ButtonRightDown = true;
     xInit = LOWORD ( lParam );
     yInit = HIWORD ( lParam );
     return 0;                        // Jump Back
    }

    case WM_RBUTTONUP :
    {
     ReleaseCapture();
     ButtonRightDown = false;
     return 0;                        // Jump Back
    }

    case WM_MOUSEMOVE :
    {
     EvMouseMove(LOWORD ( lParam ) /*x*/,
                 HIWORD ( lParam ) /*y*/,
                 xInit, yInit,
                 ButtonLeftDown, ButtonRightDown);
     return 0;                                          // Jump Back
    }

    case WM_PAINT:
    {
     PAINTSTRUCT ps;
     BeginPaint(hWnd, &ps);
     if (hRC)
     {
       Display();
     }
     EndPaint(hWnd, &ps);
     return 0;
    }

    case WM_TIMER :
    {
     int idTimer = LOWORD (wParam);
     Animation(idTimer);
     return 0;
    }
  }

  // Pass All Unhandled Messages To DefWindowProc
  return DefWindowProc(hWnd,uMsg,wParam,lParam);
}

//******************************************************************************
// Point d'entrée du programme : WinMain
//******************************************************************************
int WINAPI WinMain(HINSTANCE /*hInstance*/,     // Instance
            HINSTANCE /*hPrevInstance*/,        // Previous Instance
            LPSTR     /*lpCmdLine*/,           // Command Line Parameters
            int       /*nCmdShow*/)            // Window Show State
{
  MSG    msg;                                   // Windows Message Structure
```

```
  // Ask The User Which Screen Mode They Prefer
  if (MessageBox(NULL,"Would You Like To Run In Fullscreen Mode?",
                 "Start FullScreen?",MB_YESNO|MB_ICONQUESTION)==IDNO)
  {
    fullscreen=FALSE;                  // Windowed Mode
  }

  // Create Our OpenGL Window
  if (!CreateGLWindow("Canevas OpenGL avec l'API Window",
                                     800, 600, 16, fullscreen))
  {
    return 0;                          // Quit If Window Was Not Created
  }

  // process messages
  // Quand l'application envoie le message WM_QUIT, GetMessage()
  // renvoie la valeur 0
  // Si il y a une erreur, GetMessage() renvoie une valeur négative.
  while (GetMessage(&msg, NULL, 0, 0) > 0)
  {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
  }

  // Shutdown
  KillTimer(hWnd, ID_TIMER);
  KillGLWindow();                // Kill The Window
  return (msg.wParam);           // Exit The Program
}
```

```
// Fichier Initialisation.cpp

#include <windows.h>             // Header File For Windows
#include "MesRessources.rh"
#include "globalwin.h"
#include "evenements.h"
#include "principal.h"
#include "initialisationgl.h"
#include "initialisation.h"

/*
 * This Code Creates Our OpenGL Window.  Parameters Are:                    *
 * title      - Title To Appear At The Top Of The Window                    *
 * width      - Width Of The GL Window Or Fullscreen Mode                   *
 * height     - Height Of The GL Window Or Fullscreen Mode                  *
 * bits       - Number Of Bits To Use For Color (8/16/24/32)                *
 * fullscreenflag  - Use Fullscreen Mode (TRUE) Or Windowed Mode (FALSE)    *
 */

BOOL CreateGLWindow(char* title, int width, int height,
                    int bits, bool fullscreenflag)
{
  if(fullscreenflag)
  {
    width = GetSystemMetrics(SM_CXSCREEN);
    height = GetSystemMetrics(SM_CYSCREEN  );
  }
  int PixelFormat;      // Holds The Results After Searching For A Match
  WNDCLASSEX wc;        // Windows Class Structure
  DWORD dwExStyle;      // Window Extended Style
  DWORD dwStyle;        // Window Style
```

```
RECT WindowRect;     // Grabs Rectangle Upper Left / Lower Right Values
WindowRect.left   = (long)0;       // Set Left Value To 0
WindowRect.right  = (long)width;   // Set Right Value To Requested Width
WindowRect.top    = (long)0;       // Set Top Value To 0
WindowRect.bottom = (long)height;  // Set Bottom Value To Requested Height

fullscreen = fullscreenflag;       // Set The Global Fullscreen Flag

hInstance = GetModuleHandle(NULL);  // Grab An Instance For Our Window
wc.cbSize        = sizeof(WNDCLASSEX);
wc.style         = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;// Redraw On Size,
                                                    // And Own DC For Window.
wc.lpfnWndProc   = (WNDPROC) WndProc;   // WndProc Handles Messages
wc.cbClsExtra    = 0;                   // No Extra Window Data
wc.cbWndExtra    = 0;                   // No Extra Window Data
wc.hInstance     = hInstance;           // Set The Instance
wc.hIcon         = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_MYICON));
                   //IDI_WINLOGO);      // Load The Default Icon
wc.hIconSm       = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_MYICON));
                   //IDI_APPLICATION);
wc.hCursor       = LoadCursor(NULL, IDC_ARROW);   // Load The Arrow Pointer
wc.hbrBackground = NULL;                      // No Background Required For GL
wc.lpszMenuName  = MAKEINTRESOURCE(MENU_0);  // We Want A Menu
wc.lpszClassName = "OpenGL";                 // Set The Class Name

if (!RegisterClassEx(&wc))           // Attempt To Register The Window Class
{
  MessageBox(NULL, "Failed To Register The Window Class.",
             "ERROR", MB_OK|MB_ICONEXCLAMATION);
  return FALSE;                                 // Return FALSE
}

if (fullscreen)                               // Attempt Fullscreen Mode?
{
  DEVMODE dmScreenSettings;                   // Device Mode
  // Makes Sure Memory's Cleared
  memset(&dmScreenSettings,0,sizeof(dmScreenSettings));
  // Size Of The Devmode Structure
  dmScreenSettings.dmSize=sizeof(dmScreenSettings);
  dmScreenSettings.dmPelsWidth= 0;//width;     // Selected Screen Width
  dmScreenSettings.dmPelsHeight  = 0;//height; // Selected Screen Height
  dmScreenSettings.dmBitsPerPel  = bits;       // Selected Bits Per Pixel
  dmScreenSettings.dmFields=DM_BITSPERPEL|DM_PELSWIDTH|DM_PELSHEIGHT;

  // Try To Set Selected Mode And Get Results.
  // NOTE: CDS_FULLSCREEN Gets Rid Of Start Bar.
  if (ChangeDisplaySettings(&dmScreenSettings,
                            CDS_FULLSCREEN)!=DISP_CHANGE_SUCCESSFUL)
  {
    // If The Mode Fails, Offer Two Options.  Quit Or Use Windowed Mode.
    if (MessageBox(NULL,
         "The Requested Fullscreen Mode Is Not Supported By\n
         "Your Video Card. Use Windowed Mode Instead?", "Fullscreen Mode",
         MB_YESNO|MB_ICONEXCLAMATION) == IDYES)
    {
      fullscreen = FALSE;     // Windowed Mode Selected.  Fullscreen = FALSE
    }
    else
    {
      // Pop Up A Message Box Letting User Know The Program Is Closing.
      MessageBox(NULL,"Program Will Now Close.","ERROR",MB_OK|MB_ICONSTOP);
```

```
        return FALSE;                    // Return FALSE
      }
    }
  }

  if (fullscreen)                          // Are We Still In Fullscreen Mode?
  {
    dwExStyle=WS_EX_APPWINDOW;             // Window Extended Style
    dwStyle=WS_POPUP;                      // Windows Style
//  ShowCursor(FALSE);                     // Hide Mouse Pointer
  }
  else
  {
    dwExStyle=WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;  // Window Extended Style
    dwStyle=WS_OVERLAPPEDWINDOW;                   // Windows Style
  }

  // Adjust Window To True Requested Size
  AdjustWindowRectEx(&WindowRect, dwStyle, FALSE, dwExStyle);

  // Create The Window
  if (!(hWnd=CreateWindowEx(dwExStyle,    // Extended Style For The Window
        "OpenGL",                         // Class Name
        title,                            // Window Title
        dwStyle |                         // Defined Window Style
        WS_CLIPSIBLINGS |                 // Required Window Style
        WS_CLIPCHILDREN,                  // Required Window Style
        0, 0,                             // Window Position
        WindowRect.right-WindowRect.left,// Calculate Window Width
        WindowRect.bottom-WindowRect.top,// Calculate Window Height
        NULL,                             // No Parent Window
        NULL,                  // Le menu de la classe de fenêtre est utilisé
        hInstance,                         // Instance
        NULL)))                            // Dont Pass Anything To WM_CREATE
  {
    KillGLWindow();                       // Reset The Display
    MessageBox(NULL,"Window Creation Error.","ERROR",MB_OK|MB_ICONEXCLAMATION);
    return FALSE;                         // Return FALSE
  }

  // pfd Tells Windows How We Want Things To Be
  static PIXELFORMATDESCRIPTOR pfd = {
    sizeof(PIXELFORMATDESCRIPTOR),   // Size Of This Pixel Format Descriptor
    1,                               // Version Number
    PFD_DRAW_TO_WINDOW |             // Format Must Support Window
    PFD_SUPPORT_OPENGL |             // Format Must Support OpenGL
    PFD_DOUBLEBUFFER,                // Must Support Double Buffering
    PFD_TYPE_RGBA,                   // Request An RGBA Format
    (BYTE)bits,                      // Select Our 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,                      // Accumulation Bits Ignored
    16,                              // 16Bit Z-Buffer (Depth Buffer)
    0,                               // No Stencil Buffer
    0,                               // No Auxiliary Buffer
    PFD_MAIN_PLANE,                  // Main Drawing Layer
    0,                               // Reserved
    0, 0, 0                          // Layer Masks Ignored
  };
```

```
  if (!(hDC=GetDC(hWnd)))            // Did We Get A Device Context?
  {
    KillGLWindow();                  // Reset The Display
    MessageBox(NULL,"Can't Create A GL Device Context.",
                        "ERROR",MB_OK|MB_ICONEXCLAMATION);
    return FALSE;                    // Return FALSE
  }

  // Did Windows Find A Matching Pixel Format?
  if (!(PixelFormat=ChoosePixelFormat(hDC,&pfd)))
  {
    KillGLWindow();                  // Reset The Display
    MessageBox(NULL,"Can't Find A Suitable PixelFormat.",
                "ERROR",MB_OK|MB_ICONEXCLAMATION);
    return FALSE;                   // Return FALSE
  }

  // Are We Able To Set The Pixel Format?
  if(!SetPixelFormat(hDC,PixelFormat,&pfd))
  {
    KillGLWindow();                  // Reset The Display
    MessageBox(NULL,"Can't Set The PixelFormat.",
                        "ERROR",MB_OK|MB_ICONEXCLAMATION);
    return FALSE;                   // Return FALSE
  }

  if (!(hRC=wglCreateContext(hDC)))  // Are We Able To Get A Rendering Context?
  {
    KillGLWindow();                  // Reset The Display
    MessageBox(NULL,"Can't Create A GL Rendering Context.",
                        "ERROR",MB_OK|MB_ICONEXCLAMATION);
    return FALSE;                    // Return FALSE
  }

  if(!wglMakeCurrent(hDC,hRC))       // Try To Activate The Rendering Context
  {
    KillGLWindow();                  // Reset The Display
    MessageBox(NULL,"Can't Activate The GL Rendering Context.",
                        "ERROR",MB_OK|MB_ICONEXCLAMATION);
    return FALSE;                   // Return FALSE
  }

  ShowWindow(hWnd,SW_SHOW);          // Show The Window
  SetForegroundWindow(hWnd);         // Slightly Higher Priority
  SetFocus(hWnd);                    // Sets Keyboard Focus To The Window
  ReSizeGLScene(width, height);      // Set Up Our Perspective GL Screen

  if (!InitGL())                     // Initialize Our Newly Created GL Window
  {
    KillGLWindow();                  // Reset The Display
    MessageBox(NULL,"Initialization Failed.","ERROR",MB_OK|MB_ICONEXCLAMATION);
    return FALSE;                    // Return FALSE
  }

  return TRUE;                       // Success
}
```

```
void KillGLWindow(void)                // Properly Kill The Window
{
  if (fullscreen)                      // Are We In Fullscreen Mode?
  {
    ChangeDisplaySettings(NULL,0);     // If So Switch Back To The Desktop
    ShowCursor(TRUE);                  // Show Mouse Pointer
  }

  if (hRC)                             // Do We Have A Rendering Context?
  {
    // Are We Able To Release The DC And RC Contexts?
    if (!wglMakeCurrent(NULL,NULL))
    {
      MessageBox(NULL,"Release Of DC And RC Failed.",
                 "SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
    }

    if (!wglDeleteContext(hRC))        // Are We Able To Delete The RC?
    {
      MessageBox(NULL,"Release Rendering Context Failed.",
                 "SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
    }
    hRC=NULL;                          // Set RC To NULL
  }

  if (hDC && !ReleaseDC(hWnd,hDC))     // Are We Able To Release The DC
  {
    MessageBox(NULL,"Release Device Context Failed.",
                 "SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
    hDC=NULL;                          // Set DC To NULL
  }

  if (hWnd && !DestroyWindow(hWnd))    // Are We Able To Destroy The Window?
  {
    MessageBox(NULL,"Could Not Release hWnd.",
                 "SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
    hWnd=NULL;                        // Set hWnd To NULL
  }

  if (!UnregisterClass("OpenGL",hInstance))  // Are We Able To Unregister Class
  {
    MessageBox(NULL,"Could Not Unregister Class.",
                 "SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
    hInstance=NULL;                      // Set hInstance To NULL
  }
}
```

```
// Fichier GlobalWin.cpp

#include <windows.h>

bool fullscreen=TRUE; // Fullscreen Flag Set To Fullscreen Mode By Default
HDC hDC=NULL;         // Private GDI Device Context
HGLRC hRC=NULL;       // Permanent Rendering Context
HWND hWnd=NULL;       // Holds Our Window Handle

bool active=TRUE;     // Window Active Flag Set To TRUE By Default

HINSTANCE hInstance;  // Holds The Instance Of The Application
```

```
// Fichier GlobalWin.h

#ifndef GLOBALWIN_H
#define GLOBALWIN_H

extern bool fullscreen;
extern HINSTANCE hInstance;    // Holds The Instance Of The Application
extern HDC hDC;                // Private GDI Device Context
extern HGLRC hRC;              // Permanent Rendering Context
extern HWND hWnd;              // Holds Our Window Handle

extern bool active;            // Window Active Flag Set To TRUE By Default

#endif
```

```
// Fichier Global.cpp

#include <gl\gl.h>      // Header File For The OpenGL32 Library
#include <gl\glu.h>       // Header File For The GLu32 Library
#include "global.h"

int Mode = 0x3;  // GL_SMOOTH, GL_LINE, DEPTH, CULL
extern const int bCull = 0x1;        // 00000001
extern const int bDepth = 0x2;       // 00000010
extern const int bOutline = 0x4;     // 00000100
extern const int bShadeModel = 0x8;   // 00001000

GLfloat angle = 0.0f;
bool animationOn = true;
GLfloat xRot = 0.0f;
GLfloat yRot = 0.0f;
GLfloat Theta = 90.0f;
GLfloat xPos = 0.0f;
GLfloat yPos = 2.0f;
GLfloat zPos = 15.0f;

// Light values and coordinates
// w = 1 --> Position = (x,y,z)
// w = 0 --> Position = infini
GLfloatlightPos[] = { 10.0f, 5.0f, 5.0f, 1.0f };
GLfloat ambientLight[] = { 0.2f, 0.2f, 0.2f, 1.0f};
GLfloat diffuseLight[] = { 1.0f, 1.0f, 1.0f, 1.0f};
GLfloat specularLight[] = { 1.0f, 1.0f, 1.0f, 1.0f};
GLfloat RotationLumiere = 0.0;
```

```
// Fichier Global.h

#ifndef GLOBAL_H
#define GLOBAL_H

#define ID_TIMER 1

extern GLfloat angle;
extern bool animationOn;
extern GLfloat xRot;
extern GLfloat yRot;
extern GLfloat Theta;
extern GLfloat xPos;
extern GLfloat yPos;
extern GLfloat zPos;
```

```
extern int Mode;  // GL_SMOOTH, GL_LINE, DEPTH, CULL
extern const int bCull;
extern const int bDepth;
extern const int bOutline;
extern const int bShadeModel;

// Light values and coordinates
extern GLfloat lightPos[];
extern GLfloat ambientLight[];
extern GLfloat diffuseLight[];
extern GLfloat specularLight[];
extern GLfloat RotationLumiere;

#endif
```

```
// Fichier InitialisationGL.cpp

#include <gl\gl.h>      // Header File For The OpenGL32 Library
#include <gl\glu.h>       // Header File For The GLu32 Library

int InitGL(void)                        // All Setup For OpenGL Goes Here
{
  glShadeModel(GL_SMOOTH);                // Enable Smooth Shading
  glClearColor(1.0f, 1.0f, 1.0f, 1.0f);        // Black Background
  glClearDepth(1.0f);                     // Depth Buffer Setup
  glEnable(GL_DEPTH_TEST);                // Enables Depth Testing
  glDepthFunc(GL_LEQUAL);                 // The Type Of Depth Testing To Do
  // Really Nice Perspective Calculations
  glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
  return 1;                               // Initialization Went OK
}
```

```
// Fichier Evenements.cpp

#include <windows.h>    // Header File For Windows
#include <gl\gl.h>      // Header File For The OpenGL32 Library
#include <gl\glu.h>     // Header File For The GLu32 Library
#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include "global.h"
#include "globalwin.h"
#include "initialisation.h"
#include "geometrie.h"
#include "evenements.h"

#define COS(X) cos( (X) * M_PI/180.0)
#define SIN(X) sin( (X) * M_PI/180.0)

void ReSizeGLScene(int width, int height) // Resize And Initialize The GL Window
{
  if (height == 0) height = 1;            // Prevent A Divide By Zero By
  if (width == 0) width = 1;

  glViewport(0,0,width,height);           // Reset The Current Viewport

  glMatrixMode(GL_PROJECTION);            // Select The Projection Matrix
  glLoadIdentity();                       // Reset The Projection Matrix
```

```
   // Calculate The Aspect Ratio Of The Window
   gluPerspective(45.0f,(GLfloat)width/(GLfloat)height,0.1f,500.0f);

  DeplacerLaCamera(0.0f, Theta);
}

void Display(void)                    // Here's Where We Do All The Drawing
{
   // Clear Screen And Depth Buffer
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glPushMatrix();
       glRotatef(xRot, 1.0f, 0.0f, 0.0f);     // Effectuer les rotations
       glRotatef(yRot, 0.0f, 1.0f, 0.0f);
       DessinerLaScene();
   glPopMatrix();
   SwapBuffers(hDC);                          // Swap Buffers (Double Buffering)
}

void EvKeyDown(unsigned char key)
{
  static char Texte[80];
  key = (unsigned char) tolower(key);
  switch(key)
  {
  // Si la touche F1 a été pressée :
  case VK_F1 :                    // Is F1 Being Pressed?
     KillGLWindow();              // Kill Our Current Window
     fullscreen=!fullscreen;      // Toggle Fullscreen / Windowed Mode
     // Recreate Our OpenGL Window
     if (!CreateGLWindow("Canevas OpenGL avec l'API Window",
                                      800,600,16,fullscreen))
     {
        PostQuitMessage(0);       // Send A Quit Message
     }
     break;

  case VK_ESCAPE :                // Quitter l'application avec ESC
      PostQuitMessage(0);
      break;

  case ' ' :
       if (animationOn)    KillTimer(hWnd, ID_TIMER);
       animationOn = !animationOn;
       if (animationOn)
       {
         // Pour plus de précision, il faudrait utiliser timeSetEvent()
         if(!SetTimer(hWnd, ID_TIMER, 10, NULL))
           MessageBox(hWnd, "SetTimer Failed", "Error", MB_OK);
       }
       break;

  case '1' : Mode ^= 0x1; InvalidateRect(hWnd, NULL, FALSE);
             sprintf(Texte, "Mode (GL_SMOOTH, GL_LINE, DEPTH, CULL) : %X", Mode);
             SetWindowText(hWnd, Texte);
             break;
  case '2' : Mode ^= 0x2; InvalidateRect(hWnd, NULL, FALSE);
             sprintf(Texte, "Mode (GL_SMOOTH, GL_LINE, DEPTH, CULL) : %X", Mode);
             SetWindowText(hWnd, Texte);
             break;
  case '3' : Mode ^= 0x4; InvalidateRect(hWnd, NULL, FALSE);
             sprintf(Texte, "Mode (GL_SMOOTH, GL_LINE, DEPTH, CULL) : %X", Mode);
```

```
            SetWindowText(hWnd, Texte);
            break;
   case '4' : Mode ^= 0x8; InvalidateRect(hWnd, NULL, FALSE);
            sprintf(Texte, "Mode (GL_SMOOTH, GL_LINE, DEPTH, CULL) : %X", Mode);
            SetWindowText(hWnd, Texte);
            break;

   case VK_UP :
       DeplacerLaCamera(1.0f, Theta);
       InvalidateRect(hWnd, NULL, FALSE);
       break;

   case VK_DOWN :
       DeplacerLaCamera(-1.0f, Theta);
       InvalidateRect(hWnd, NULL, FALSE);
       break;

   case VK_LEFT :
       Theta += 1.0f;
       DeplacerLaCamera(0.0f, Theta);
       InvalidateRect(hWnd, NULL, FALSE);
       break;

   case VK_RIGHT :
       Theta -= 1.0f;
       DeplacerLaCamera(0.0f, Theta);
       InvalidateRect(hWnd, NULL, FALSE);
       break;
  }
}

void EvMouseMove(int x, int y, int & xInit, int & yInit,
                 bool ButtonLeftDown, bool ButtonRightDown)
{
  static int DeltaX;
  static int DeltaY;

  if(ButtonLeftDown) // Le bouton gauche est en bas
  {
    DeltaX = xInit - x;
    DeltaY = yInit - y;
    if(abs(DeltaY) > abs(DeltaX))
    {
      if(DeltaY > 0)
      {
        xRot++;
        if(xRot >= 360.0) xRot = 0.0;
      }
      else
      {
        xRot--;
        if(xRot <= -360.0) xRot = 0.0;
      }
    }
    else
    {
      if(DeltaX > 0)
      {
        yRot++;
        if(yRot >= 360.0) yRot = 0.0;
      }
```

```
      else
      {
        yRot--;
        if(yRot <= -360.0) yRot = 0.0;
      }
    }

    xInit = x;
    yInit = y;
    InvalidateRect(hWnd, NULL, FALSE);
  }

  if(ButtonRightDown)
  {
    GLfloat d = 0.0f;
    DeltaX = xInit - x;
    DeltaY = yInit - y;

    if(abs(DeltaY) > abs(DeltaX))
    {
      // Si on a déplacé la souris vers le haut
      if(DeltaY > 0)
        d = 1.0f;
      // Si on a déplacé la souris vers le bas
      else
        d = -1.0f;
    }
    else
    {
      // Si on a déplacé la souris à droite
      if (DeltaX > 0)
        Theta += 1.0f;
      // Si on a déplacé la souris à gauche
      else
        Theta -= 1.0f;
    }

    DeplacerLaCamera(d, Theta);
    xInit = x;
    yInit = y;
    InvalidateRect(hWnd, NULL, FALSE);
  }
}

void Animation(int Id)
{
  // Pour faire tourner la lumière
  if(Id == ID_TIMER)
  {
    RotationLumiere++;
    if(RotationLumiere >= 360.0) RotationLumiere = 0.0;
    InvalidateRect(hWnd, NULL, FALSE);
  }
}

void DeplacerLaCamera(GLfloat d, GLfloat Theta)
{
  static GLfloat fRadius = 50.0f;
  GLfloat xDelta, zDelta;
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();  // Charger la matrice de visualisation identité
```

```
  xDelta = d*cos(Theta*M_PI/180.0);
  zDelta = -d*sin(Theta*M_PI/180.0);
  xPos += xDelta;
  zPos += zDelta;

  gluLookAt(xPos,
            yPos,
            zPos,   // Position de la caméra
            xPos + fRadius*COS(Theta),
            yPos,
            zPos - fRadius*SIN(Theta),
            0.0f, 1.0f, 0.0f);  // Axe dirigé vers le haut
}
```