/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Includes

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "testogl.h"

#include "joyctrl.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Declarations

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

LRESULT CALLBACK WndProc (HWND hWnd, UINT message,

WPARAM wParam, LPARAM lParam);

void EnableOpenGL (HWND hWnd, HDC \*hDC, HGLRC \*hRC);

void DisableOpenGL (HWND hWnd, HDC hDC, HGLRC hRC);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* WinMain

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int WINAPI WinMain (HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int iCmdShow)

{

WNDCLASS wc;

HWND hWnd;

HDC hDC;

HGLRC hRC;

MSG msg;

BOOL bQuit = FALSE;

// float theta = 0.0f;

/\* register window class \*/

wc.style = CS\_OWNDC;

// wc.style = CS\_CLASSDC;

wc.lpfnWndProc = WndProc;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hInstance = hInstance;

wc.hIcon = LoadIcon (NULL, IDI\_APPLICATION);

wc.hCursor = LoadCursor (NULL, IDC\_ARROW);

wc.hbrBackground = CreateSolidBrush(RGB(127, 18, 127));

wc.lpszMenuName = NULL;

wc.lpszClassName = "GLSample";

RegisterClass (&wc);

/\* create main window \*/

HDC hdcMainScr = GetWindowDC(NULL);

hWnd = CreateWindow (

"GLSample", "OpenGL Sample",

WS\_CAPTION | WS\_POPUPWINDOW | WS\_VISIBLE,

0,

0,

GetDeviceCaps(hdcMainScr, HORZRES),

GetDeviceCaps(hdcMainScr, HORZRES)\*9/16,

NULL, NULL, hInstance, NULL);

/\* enable OpenGL for the window \*/

EnableOpenGL (hWnd, &hDC, &hRC);

// now the initialization of the game, and the start screen:

Initializations(60.0, (float)GetDeviceCaps(hdcMainScr, HORZRES), (float)(GetDeviceCaps(hdcMainScr, HORZRES)\*600/900));

ShowCursor(FALSE);

//when create window class for application window

/\* program main loop \*/

while (!bQuit)

{

/\* check for messages \*/

if (PeekMessage (&msg, NULL, 0, 0, PM\_REMOVE))

{

/\* handle or dispatch messages \*/

if (msg.message == WM\_QUIT)

{

bQuit = TRUE;

}

else

{

TranslateMessage (&msg);

DispatchMessage (&msg);

}

}

else

{

/\* OpenGL animation code goes here \*/

// not here, but in the main loop of WndProc()

}

}

/\* shutdown OpenGL \*/

DisableOpenGL (hWnd, hDC, hRC);

ShowCursor(TRUE);

/\* destroy the window explicitly \*/

DestroyWindow (hWnd);

return msg.wParam;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Window Procedure

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

LRESULT CALLBACK WndProc (HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

static HDC hdc;

static BOOL bTest;

static BOOL bLight;

static float fds; // delta-s displacement along the axis of view

static float fdq; // delta-s variable displacement to simulate acceleration

static float fas; // delta-s displacement iterated one-by-one in timer to simulate accelerating/ decelerating

static float fdt; // delta-theta // Azimouth deviation

static float fdf; // delta-phi // elevation deviation

static float fdh; // delta-hi // normal deviation

static double R; // the distance to the poiningn object //

static float fX, fY, fZ;

static CxCam Cam;

static float fAng; // angle of rotation for glRotatef() function for special effects:

static BOOL bMove; // engine status

static BOOL bJoyX; // command taken from joystick

static BOOL bStop; // engine stopped

static BOOL brake; // brake pressed

static BOOL bBall; // there is a ball or not on the scene...

static HFONT hFont;

static HFONT hFOld;

static POINT pM[2]; // Mouse position

LPWSTR wszMsg = (LPWSTR)GlobalAlloc(GPTR, 0xff);

switch (message)

{

case WM\_CREATE:

{

bTest = FALSE;

bLight = FALSE;

fds = 0.1f; // 10 cm

fX = 0.0f; fY = 0.0f; fZ = 0.0f;

fdt = (float)DEG; // 1 Deg.

fdf = (float)DEG; // 1 Deg.

ZeroMemory(&Cam, sizeof(CxCam));

ZeroMemory(pM, sizeof(POINT)\*2);

Cam.R = 2.0f;

Cam.x = 0.0f;

Cam.x0 = Cam.x+Cam.R;

Cam.z = 1.75f;

Cam.z0 = Cam.z; // look direct to the horizon

Cam.zv = 1.0f; // normal vector oriented just up

fAng = 0.0f;

bMove = FALSE; // engine stopped

bJoyX = FALSE; // no default control to joystick, but mouse at first

bStop = TRUE; // engine stopped

brake = FALSE; // no break pressed

bBall = FALSE; // no ball on the scene...

// R = sqrt((Cam.x0-Cam.x)\*(Cam.x0-Cam.x)+(Cam.y0-Cam.y)\*(Cam.y0-Cam.y)+(Cam.z0-Cam.z)\*(Cam.z0-Cam.z));

swprintf(wszMsg, L"R = %g (it must be 1.0)", R);

// MessageBoxW(NULL, wszMsg, L"", MB\_OK);

hdc = GetDC(hWnd);

// Print welcome string:

hFont = CreateFont(10,0,0,0,FW\_DONTCARE,FALSE,FALSE,FALSE,DEFAULT\_CHARSET,OUT\_OUTLINE\_PRECIS,

CLIP\_DEFAULT\_PRECIS,DEFAULT\_QUALITY, VARIABLE\_PITCH,TEXT("Courier"));

hFOld = (HFONT)SelectObject(hdc, hFont);

SetBkMode(hdc, TRANSPARENT);

SetTextColor(hdc, RGB(255, 255, 0));

/////// code for joystick:

// Register for joystick devices

//

RAWINPUTDEVICE rid;

rid.usUsagePage = 1;

rid.usUsage = 4; // Joystick

rid.dwFlags = 0;

rid.hwndTarget = hWnd;

if(RegisterRawInputDevices(&rid, 1, sizeof(RAWINPUTDEVICE)))

{

bJoyX = TRUE;

}

else

{

MessageBoxW(NULL, L"Joystick registering failed", L"Joystick message", MB\_OK);

}

break;

}

case WM\_ACTIVATE:

{

if(LOWORD(wParam==WA\_ACTIVE))

{

Redraw(hdc, false, &Cam);

MoveIt(&Cam, 0.0);

RotateIt(&Cam, 0.0, 0.0);

}

break;

}

case WM\_CLOSE:

{

GlobalFree(wszMsg);

SelectObject(hdc, hFOld);

ReleaseDC(hWnd, hdc);

ShowCursor(TRUE);

PostQuitMessage (0);

return 0;

}

case WM\_DESTROY:

{

ReleaseDC(hWnd, hdc);

return 0;

}

case WM\_INPUT:

{

if(bJoyX)

{

//

// Get the pointer to the raw device data, process it and update the window

//

CxJoyStk\* pJoX = (CxJoyStk\*)GlobalAlloc(GPTR, sizeof(CxJoyStk));

PRAWINPUT pRawInput;

UINT bufferSize;

HANDLE hHeap;

GetRawInputData((HRAWINPUT)lParam, RID\_INPUT, NULL, &bufferSize, sizeof(RAWINPUTHEADER));

hHeap = GetProcessHeap();

pRawInput = (PRAWINPUT)HeapAlloc(hHeap, 0, bufferSize);

if(!pRawInput)

return 0;

GetRawInputData((HRAWINPUT)lParam, RID\_INPUT, pRawInput, &bufferSize, sizeof(RAWINPUTHEADER));

ParseRawInput(pRawInput, pJoX);

HeapFree(hHeap, 0, pRawInput);

// InvalidateRect(hWnd, NULL, TRUE);

// UpdateWindow(hWnd);

PrintJoystickCoords(pJoX, hdc);

if(bMove) // replace with a condition like activating the joystick

{

fds = (float)pJoX->lAxisY/128;

fds = -fds;

if(fds>+0.9) fds=+0.9;

if(fds<-0.9) fds=-0.9;

}

if(bMove) // replace with a condition like activating the joystick

{

fdt = (float)pJoX->lAxisX\*0.1/DEG;

fdt = -fdt;

if(fdt>+35\*DEG) fdt=+35\*DEG;

if(fdt<-35\*DEG) fdt=-35\*DEG;

}

if(pJoX->lHat==1)

{

RotateIt(&Cam, 0.0, +fdf\*0.2);

// Redraw(hdc, bLight, pCam); // redraw will be automatically done by the timer...

}

if(pJoX->lHat==5)

{

RotateIt(&Cam, 0.0, -fdf\*0.2);

// Redraw(hdc, bLight, pCam); // redraw will be automatically done by the timer...

}

GlobalFree(pJoX);

}

break;

}

case WM\_RBUTTONDOWN:

{

// MessageBoxW(NULL, L"here", L"", MB\_OK);

TestBitmap(hdc);

break;

}

case WM\_MOUSEWHEEL:

{

if(bMove)

{

float fD = GET\_WHEEL\_DELTA\_WPARAM(wParam)/120.0/50;

switch(LOWORD(wParam))

{

case MK\_SHIFT:

{

Cam.y+=fD;

Cam.y0+=fD;

MoveIt(&Cam, 0.0);

break;

}

case MK\_CONTROL:

{

Cam.z+=fD;

Cam.z0+=fD;

MoveIt(&Cam, 0.0);

break;

}

default:

{

fdf = fD/5.0;

RotateIt(&Cam, 0.0, fdf);

break;

}

}

}

break;

}

case WM\_MOUSEMOVE:

{

if(!bJoyX)

{

if(bMove)

{

pM[1].x = LOWORD(lParam); // read the new position of the mouse to give the direction to move

pM[1].y = HIWORD(lParam);

if((abs(pM[1].y-pM[0].y))>10) // y for mouse is x for the scene!!!

{

fds = (pM[1].y-pM[0].y-10)/250.0;

fds = -fds;

if(fds>+0.9) fds=+0.9;

if(fds<-0.9) fds=-0.9;

}

else

{

fds = 0.0;

}

if((abs(pM[1].x-pM[0].x))>(abs(fds)<0.01)\*10) // x for mouse is y for the scene!!!

{

fdt = (pM[1].x-pM[0].x-10)/25.0/DEG;

fdt = -fdt;

if(fdt>+5\*DEG) fdt=+5\*DEG;

if(fdt<-5\*DEG) fdt=-5\*DEG;

}

else

{

fdt = 0.0;

}

}

else

{

pM[0].x = LOWORD(lParam); // conserve the las position of the mouse when engine is not started

pM[0].y = HIWORD(lParam);

}

}

break;

}

case WM\_TIMER:

{

switch(wParam)

{

case ID\_TIMER:

{

if(fdt) // if direction...

{

RotateIt(&Cam, +fdt, 0.0);

}

if(fds+fdq)

{

if(!bStop) // accelerating effect by moving along

fdq+=(fds-fdq)\*0.05; // speed ispoint following the setpoint

else // decelerating

fdq\*=0.95; // speed ispoint decreasing till zero...

MoveIt(&Cam, fdq);

}

if(bStop) // if engine stopped by user ...

{

if((fdq>-0.005)&&(fdq<+0.005)) // if engine is stopped and no more speed

{

KillTimer(hWnd, ID\_TIMER); // Timer to move...

fds = 0.1; // reset all to the initial values

fdq = 0.0;

fdt = DEG;

fdf = DEG;

bMove = FALSE;

}

}

if(brake) // ... as long as the brake is pressed

{

fds = 0.0; // speed setpoint to zero

fdq\*=0.75; // speed ispoint decreasing

// condition for fdq....???>

fdt = 0.0; // no angle movements

fdf = 0.0;

}

Redraw(hdc, bLight, &Cam);

PrintCoords(&Cam, hdc);

PrintParams(fds, fdq, fdt, fdf, hdc);

break;

}

case ID\_TIMES:

{

// SetFAng(fAng++); // \_\_???

break;

}

}

break;

}

case WM\_KEYUP:

{

switch(wParam)

{

case VK\_SPACE:

{

brake = FALSE; // brake released

pM[0].x = pM[1].x;

pM[0].y = pM[1].y; // in order to not to have to readuce the mouse in the zero-position

break;

}

}

break;

}

case WM\_KEYDOWN:

switch (wParam)

{

case VK\_RETURN:

{

// MoveIt(&Cam, +fds);

Redraw(hdc, bLight, &Cam);

PrintCoords(&Cam, hdc);

// Rotation effects of some elements (cubes)

// SetTimer(hWnd, ID\_TIMES, 50, NULL);

// MoveIt(&Cam, 0.0);

break;

}

case VK\_ESCAPE: // terminating the application

{

GlobalFree(wszMsg);

SelectObject(hdc, hFOld);

ReleaseDC(hWnd, hdc);

ShowCursor(TRUE);

PostQuitMessage (0);

return 0;

}

case VK\_INSERT: // starting the engines

{

// starting effect:

// end of starting effect

bStop = FALSE;

if(!bMove)

{

SetTimer(hWnd, ID\_TIMER, 50, NULL);

fds = 0.0; // set all to zero, in order to be set again from the mouse position

fdq = 0.0; // and the start value of the speed on acceleration

fdt = 0.0; // and the increment value of the angle

}

bMove = TRUE;

break;

}

case VK\_DELETE: // engine stop

{

// first stop attenuately:

bStop = TRUE;

bMove = FALSE;

break;

}

case VK\_SPACE: // brake triggered

{

brake = TRUE;

break;

}

case VK\_UP: // move forward alongside the eye==>pointing direction

{

MoveIt(&Cam, +fds);

Redraw(hdc, bLight, &Cam);

PrintCoords(&Cam, hdc);

break;

}

case VK\_DOWN: // move backwards

{

MoveIt(&Cam, -fds);

Redraw(hdc, bLight, &Cam);

PrintCoords(&Cam, hdc);

break;

}

case VK\_RIGHT: // turn right

{

if(GetKeyState(VK\_SHIFT)<0)

{

Cam.y-=fdt; Cam.y0-=fdt;

}

else

{

// RotateIt(&Cam, -fdt, 0.0);

TurnIt(&Cam, -fdt, 0.0, 0.0);

}

Redraw(hdc, bLight, &Cam);

PrintCoords(&Cam, hdc);

break;

}

case VK\_LEFT: // turn left

{

if(GetKeyState(VK\_SHIFT)<0)

{

Cam.y+=fdt; Cam.y0+=fdt;

}

else

{

// RotateIt(&Cam, +fdt, 0.0);

TurnIt(&Cam, +fdt, 0.0, 0.0);

}

Redraw(hdc, bLight, &Cam);

PrintCoords(&Cam, hdc);

break;

}

case VK\_PRIOR: // turn up

{

if(GetKeyState(VK\_SHIFT)<0)

{

Cam.z+=fdf; Cam.z0+=fdf;

}

else

{

RotateIt(&Cam, 0.0, +fdf);

}

Redraw(hdc, bLight, &Cam);

PrintCoords(&Cam, hdc);

break;

}

case VK\_NEXT: // turn down

{

if(GetKeyState(VK\_SHIFT)<0)

{

Cam.z-=fdf; Cam.z0-=fdf;

}

else

{

RotateIt(&Cam, 0.0, -fdf);

}

Redraw(hdc, bLight, &Cam);

PrintCoords(&Cam, hdc);

break;

}

case VK\_F2: // mouse/ joystick control switch

{

if(bStop)

{

if(bJoyX)

{

MessageBoxW(NULL, L"Mouse control", L"Input option changed", MB\_OK);

}

else

{

MessageBoxW(NULL, L"Stick control", L"Input option changed", MB\_OK);

}

bJoyX = !bJoyX;

}

break;

}

case VK\_F3: // decrease focus distance

{

Cam.R-=1; // haves absolutely no effect

Redraw(hdc, bLight, &Cam);

PrintCoords(&Cam, hdc);

break;

}

case VK\_F4: // increase focus distance

{

Cam.R+=1; // haves absolutely no effect!

Redraw(hdc, bLight, &Cam);

PrintCoords(&Cam, hdc);

break;

}

case VK\_HOME: // back to the origin

{

if(bStop)

{

int nOpt = MessageBoxW(NULL, L"Would you like to return to the start?", L"Back to the start", MB\_YESNO);

if(nOpt==IDYES)

{

ZeroMemory(&Cam, sizeof(CxCam));

Cam.R = 2;

Cam.x = 0.0;

Cam.x0 = Cam.x+Cam.R;

Cam.z = 1.75;

Cam.z0 = Cam.z; // look direct to the horizon

Cam.zv = 1.0; // for the moment, constant

MoveIt(&Cam, +fds);

Redraw(hdc, bLight, &Cam);

PrintCoords(&Cam, hdc);

}

}

break;

}

break;

}

return 0;

default:

{

return DefWindowProc (hWnd, message, wParam, lParam);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Enable OpenGL

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void EnableOpenGL (HWND hWnd, HDC \*hDC, HGLRC \*hRC)

{

PIXELFORMATDESCRIPTOR pfd;

int iFormat;

/\* get the device context (DC) \*/

\*hDC = GetDC (hWnd);

/\* set the pixel format for the DC \*/

ZeroMemory (&pfd, sizeof (pfd));

pfd.nSize = sizeof (pfd);

pfd.nVersion = 1;

pfd.dwFlags = PFD\_DRAW\_TO\_WINDOW|PFD\_SUPPORT\_OPENGL|PFD\_DOUBLEBUFFER;

pfd.iPixelType = PFD\_TYPE\_RGBA;

pfd.cColorBits = 24;

pfd.cDepthBits = 16;

pfd.iLayerType = PFD\_MAIN\_PLANE;

iFormat = ChoosePixelFormat (\*hDC, &pfd);

SetPixelFormat (\*hDC, iFormat, &pfd);

/\* create and enable the render context (RC) \*/

\*hRC = wglCreateContext( \*hDC );

wglMakeCurrent( \*hDC, \*hRC );

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Disable OpenGL

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void DisableOpenGL (HWND hWnd, HDC hDC, HGLRC hRC)

{

wglMakeCurrent (NULL, NULL);

wglDeleteContext (hRC);

ReleaseDC (hWnd, hDC);

}