Courses 3DCx, C1: OpenGL, Code Comments

In `CChildView.h` in front of `class CChildView : public CWnd`

`#define nMax 100` //nMax is the upper limit of the no. of vertices that can be stored. You can set nMax to 1000 or even higher. It is just a vaste of memory to reserve space for too much vertices. nMax does not influence the velocity of the animation as long as you do not fill its space by drawing.

`#include < gl/gl.h >` //This is the important OpenGL-Include file containing all function declarations of OpenGL. When your compiler does not contain this file (usually in the path `C:\Programs\Microsoft Visual Studio\VC98\Include\GL`), you have no chance to compile this project.

In `CChildView.h` in `class CChildView : public CWnd`

Declaration of private variables and data types that can be accessed by all member functions of class `CChildView`.

`CPoint old_vertex;` // to store one vertex

`CPoint p[nMax];` //to store nMax vertices

`typedef struct { float x; float y; float z; } FPoint;` //type for 2D vector graphics coordinates

`FPoint f[nMax];` //to store nMax vertices in float

`int n;` //n will be the counter of vertices.

`HGLRC hglrc;` //handle for an OpenGL-Rendering-Context (see the FAQs)

`BOOL done;` //This flag indicates that the mouse drawing is over. It is initialized to `false` in `CChildView::OnCreate()` and `CChildView::OnLButtonDown()` and changed to `true` in `CChildView::OnLButtonUp()`

In `CMainFrame::PreCreateWindow(...)`

This is a function in the class `CMainFrame` presetting the style and size of our first window.

`cs.cx = 500;` //Sets the initial window width to 500 pixels. Later we will remap the mouse coordinates in logical OpenGL coordinates where x is a float between -1.0 and + 1.0.

`cs.cy = 500;` //Sets the initial window height to 500 pixels. Later we will remap the mouse coordinates in logical OpenGL coordinates where y is a float between -1.0 and + 1.0. It is important that our client area is nearly quadratic. When `cs.cx` is heavily different from `cs.cy` we will obtain disturbing distortions.

In `CChildView::OnCreate(LPCREATESTRUCT lpCreateStruct)`

`if (CWnd ::OnCreate(lpCreateStruct) == -1) return -1;` //This line of code is from Microsoft. Leave it here, it produces an error if something goes wrong during the first construction of our client area.

`CClientDC dc( this );` //Device context of the current client area. Such device contexts normally support differnt pixel formats.

`PIXELFORMATDESCRIPTOR pfd = { sizeof( PIXELFORMATDESCRIPTOR ), 1, PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER, PFD_TYPE_RGBA, 24,0,0,0,0,0,0,0,0,0,0,0,32,0,0,PFD_MAIN_PLANE,0,0,0 };` //pfd specifies what we want from our device context (see FAQs). The most important parameter is the bit-combination of `PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER.` When we do not set these bits, we loose the opportunity for swapping a double output buffer resulting in severe flicker effects.

`int i = ChoosePixelFormat( dc.m_hDC, &pfd );` //Checks all possible pixel formats that our device context offers. Returns the no. of the one that matches best.

`SetPixelFormat( dc.m_hDC, i, &pfd );` //Realizes the best matching pixel format inside our device context.

`hglrc = wglCreateContext( dc.m_hDC );` //Makes an hardware-independent OpenGL rendering context that matches as good as possible to the current hardware- and Windows-dependent device context and returns a handle to it (or a NULL when that is not possible). The rendering context frees the OpenGL-code from any dependency from the hardware and from the operating system. Any code lines that will use this `hglrc` are truely platform-independent.

`done = FALSE;` //Ignore the timer events during drawing.

`SetTimer(1, 1, NULL);` //Start a timer no. 1 with the fastest possible velocity. Parmaeter no. 1 is the user no of this timer (64 parallel timers are possible). Parameter 2 requests an event every 1 msec. The operating system cannot react with 1000 events in a second but it will now produce such events as often as possible. Parameter 3 is not used.

`return 0;` //Exit with the normal return value of 0.

In `void CChildView::OnPaint()`

`CPaintDC dc(this);`//Device context of the current client area

`dc.TextOut( 0, 0, "Press the left mouse button and move!" );` //Text for users who do not know what to do.

In `void CChildView::OnLButtonDown(UINT nFlags, CPoint point)`

`done = FALSE;` //Ignore the timer events during drawing.

`old_vertex = p[0] = point;` //Store the first vertex.

`n = 1;` //Reset the no. of vertices.

`Invalidate();` //If there was a former drawing it is swept now.

In `void CChildView::OnMouseMove(UINT nFlags, CPoint point)`

```
if (!nFlags) return; //Ignore the mouse movement if no mouse buttons are pressed.
int dx = point.x - old_vertex.x; //horizontal distance to the former vertex
int dy = point.y - old_vertex.y; //vertical distance to the former vertex
if ( dx*dx + dy*dy < 100 ) return; //If the distance is less the 10 forget the point.
if ( n > nMax - 2 ) return; //If there are already more than 98 vertices forget the rest.
CClientDC dc(this); //Device context of the current client area
dc.MoveTo( old_vertex ); dc.LineTo( point ); //Draw a straight line.
old_vertex = p[n++] = point; //Remember the current point as the latest one.
```

In `void CChildView::OnLButtonUp(UINT nFlags, CPoint point)`

The drawing is done. The animation must be prepared.

```
CRect r; GetClientRect( r ); //We need the current extend of the client area.
int i, xnew, ynew; //local intermediary integer variables
int halfwidth = r.Width ()/2; //mid between left and right border of the client area
int halfheight = r.Height()/2; //mid between top and bottom of the client area
for ( int i = 0; i < n; i++ ) //Take any vertex
xnew = p[i].x - halfwidth;//Scroll left. Any x left of the midpoint comes out negative.
ynew = p[i].y - halfheight;//Scroll up. Any y above of the midpoint comes out negative.
ynew *= -1; //Invert the Y-axis upside up. Any y above of the midpoint comes out positive.
f[i].x = float(xnew) / float(halfwidth); //Convert to float and scale between -1.f and +1.f.
f[i].y = float(ynew) / float(halfheight); //Convert to float and scale between -1.f and +1.f.
done = TRUE; //The animation can start now.
```

In `void CChildView::OnTimer(UINT nIDEvent)`

`if ( !done ) return;` //Do nothing during the time between `OnLButtonDown` and `OnLButtonUp.`

`int i;` //local help variable

`CClientDC dc( this );` //Device context of the current client area

`wglMakeCurrent( dc.m_hDC, hglrc );` //Windows is informed, that it has to map in the future the `hglrc` rendering context onto its current `dc.m_hDC` device context. All graphics drawn by OpenGl into its rendering context are accepted from now on by Windows to be automatically drawn into the device context of the currently active window (more exactly: into the device context of the client area of the currently active window, which is ours in this moment).

`glDrawBuffer( GL_BACK );` //Do not draw into the visible buffer. In order to prevent the graphics from flickering draw into the invisible back buffer.

`glMatrixMode( GL_MODELVIEW );` //The following statement `glRotatef` shall rotate the polygon model and not the observation point nor the texture.

`glRotatef( 2.0f, 1.f, 1.f, 1.f );` //Prepare a 2 degree rotation around a vector pointing from 0/0/0 to 1/1/1. This vector is at 45 degrees with the X-, the Y and the Z-axis. The rotation will be executed inside the `glBegin() - glEnd()` clauses below.

`glLineWidth ( 3 );` //All following lines will be drawn with a pen of thickness 3.

`glPointSize( 9 );` //All following points will be drawn with a pen of thickness 9

`glColor3f( 0.f, 1.f, 0.f );` //The pen color is green.

`glClearColor( 1.f, 1.f, 1.f, 0.f);` //The color to erase the canvas is white without any transparency.

`glClear(GL_COLOR_BUFFER_BIT);` //Erase everything.

`glBegin( GL_LINE_LOOP );` //Start do draw a polygone.

`for ( i = 0; i < n; i++ )` //Take all 3D-vertices.

`glVertex3f( f[i].x, f[i].y, f[i].z );` //Draw straight 3D-lines from vertex 0 to vertex n-1.

`glEnd();` //Stop drawing.

`glBegin( GL_POINTS );` //Start to draw points

`for ( i = 0; i < n; i++ )` //Take all 3D-vertices.

`glVertex3f ( f[i].x, f[i].y, f[i].z );` //Mark the vertex i by a thick rectangular point.

`glEnd();` //Stop drawing.

`SwapBuffers( dc.m_hDC );` //Show the background buffer on the screen, hide the foreground buffer and exchange their names.

`dc.Polyline( p, n );` //Draw a thin black 2D-polygone without any OpenGL just using the Windows API and the Windows DC. This polygone will flicker, because it does not use the double buffer mechanism of OpenGL. This statement is just for demonstration that Windows graphics and OpenGL graphics can be used in parallel without disturbing each other.