Courses 2DCx, C2: Draw, Code Comments

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

`private: CPoint old_vertex;` //The key word `private:` tells the compiler that the variable that follows is just for local use by `CChildView`. `CPoint` is a simple MFC class consisting of two integers x and y. `CPoint` is the ideal data type to store and handle mouse coordinates. The variable `old_vertex` will be used to store the old mouse x,y during mouse movements while the program is waiting for a new x,y that will replace `old_vertex`.
Vertex (plural: vertices) is the latin word for edge. In 2D computer graphics vertex means a pair of x and y coordinates, in 3D a triple of x, y and z coordinates.

In `CChildView.cpp` inside `void CChildView::OnLButtonDown(...)`

`old_vertex = point; Invalidate();` //Whenever the user of the program presses his left mouse button, the function `OnLButtonDown` is called. The first parameter `nFlags` transports the information if other mouse buttons and/or if the Ctrl- and Shift-keys on the keyboard have been pressed at the same time. The second parameter `point` transports the mouse coordinate. We ignore the first parameter and store the second into our `private` variable `old_vertex`. The next statement `Invalidate()` asks the operating system to completly erase all former drawings from our window.

In `CChildView.cpp` inside `void CChildView::OnMouseMove(...)`

`if ( !nFlags ) return;` //Whenever the mouse moves (even unvoluntarily when the table trembles slightly), the function `OnMouseMove` is called. In order to avoid unvoluntary drawing we check the parameter nFlags. If it is zero, no mouse button has been pressed and we quit the function and forget about this mouse movement.

`CClientDC dc(this);` //The graphic statements `MoveTo` and `LineTo` need a Device Context to draw into. The statement `dc(this)` returns a reference to the Device Context of the currently active window.

`dc.MoveTo( old_vertex ); dc.LineTo( point );` //These two statements draw a digital line between two points.

`old_vertex = point;` //We forget the content of `old_vertex` and replace it by the current coordinate. The next line will start from here.

In `CChildView.cpp` inside `void CChildView::OnPaint()`

`dc.TextOut( 0, 0, "Press the left mouse button and move!" );` //This text remembers the user that the program is waiting for mouse events.

In `CChildView.cpp` inside `void CChildView::OnMouseMove(...)`

`CString blabla;` //We need a local string that can be filled with values.

`blabla.Format( "x=%d, y=%d ", point.x, point.y );` //Fill it with two integers i.e. the current x and y.

`dc.TextOut( 0, 20, blabla );` //Display this line 20 pixels below the first text line

In Version 3 `CChildView.cpp` inside `void CChildView::OnMouseMove(...)`

`dc.Rectangle( point.x-3, point.y-3, point.x+3, point.y+3 );` //Draw a 7x7 square around every vertex.

In Version 4 `CChildView.cpp` inside `void CChildView::OnMouseMove(...)`

`int dx = point.x - old_vertex.x;` //horizontal distance between `old_vertex` and the current mouse position

---

`int dy = point.y - old_vertex.y;` //vertical distance between `old_vertex` and the current mouse position

---

`if ( dx*dx + dy*dy < 100 ) return;` //If the distance is less than 10, forget the current point (Pythagoras' theorem).

In Version 5 `CChildView.h` in front of all other lines

`#define nMax 100` //nMax is now a synonym of 100

`#include < math.h >` //necessary for the declaration of the function `_hypoth` below

---

In Version 5 `CChildView.h` inside `CChildView : public CWnd`

`CPoint polygon[nMax];` //Array for 100 vertices each containing 2 integers x and y.

`CPoint center_of_gravity;` //One vertex to compute and store the center of gravity

`CRect minmax;` //`CRect` is a simple MFC class to store and handle axis-parallel rectangles. Purpose of minmax: storing the minimal surrounding rectangle of the polygone.

`int n;` //simple integer to count vertices

`double perimeter, area;` //two float variables with double precision

In Version 5 `CChildView.cpp` inside `CChildView :: OnLButtonDown(...)`

`polygon[0] = old_vertex = point;` //This is the starting vertex. Please notice the very important feature of C++: The first vertex always has number zero.

`n = 1;` //The number of vertices is now 1.

In Version 5 `CChildView.cpp` inside `CChildView :: OnMouseMove(...)`

`if ( n > nMax - 2 ) return;` //There is no memory space for more than 99 vertices.

`old_vertex = polygon[n++] = point;` //Store the current vertex into the array `polygon` and increment the vertex counter.

In Version 5 `CChildView.cpp` inside `CChildView :: OnLButtonUp(...)`

`if ( n < 2 ) return;` //If there are less than 2 vertices, nothing can be drawn.

`polygon[n++] = polygon[0];` //In order to close the drawing, copy the first vertex inside `polygon` at the end of `polygon`.

`perimeter = area = 0.;` //intialize before start of summation

`center_of_gravity.x = center_of_gravity.y = 0;` //intialize before start of summation

`minmax.left = minmax.right = polygon[0].x;` //initialize the surrounding rectangle

`minmax.top = minmax.bottom = polygon[0].y;` //initialize the surrounding rectangle

`for ( int i = 1; i < n; i++ )` //start with the second vertex and run through all vertices

---

`int x = polygon[i].x;` //It is is shorter to write x then `polygon[i].x.`

`int y = polygon[i].y;` //It is is shorter to write y then `polygon[i].y.`

`double dx = double( x - polygon[i-1].x );` //horizontal difference between two adjacent vertices

`double dy = double( y - polygon[i-1].y );` //vertical difference between two adjacent vertices

`double my = double( y + polygon[i-1].y ) * 0.5;` //mean of two adjacent y-coordinates

`perimeter += _hypot( dx, dy );` //Pythagoras' theorem

`area += dx * my;` //Trapezoid formula

`center_of_gravity.x += x;` //sum up all x-coordiantes

`center_of_gravity.y += y;` //sum up all y-coordiantes

`if ( x < minmax.left ) minmax.left = x;` //if the new vertex is on the left of the `minmax`

`if ( x > minmax.right ) minmax.right = x;` //if the new vertex is on the right of `minmax`

`if ( y < minmax.top ) minmax.top = y;` //if the new vertex is above of `minmax`

`if ( y > minmax.bottom ) minmax.bottom = y;` //if the new vertex is below of `minmax`

`center_of_gravity.x /= n-1;` //divide the x-sum by n-1. We did not sum up the first vertex polygon[0] because it is identical to the last vertex polygon[n-1] and we do not want to count it twice.

`center_of_gravity.y /= n-1;` //divide the y-sum by n-1.

`Invalidate();` //Jump to `OnPaint(),` erase all, redraw all.

In Version 5 `CChildView.cpp` inside `CChildView :: OnPaint()`

`if ( n < 2 ) return;` //There is not a single line to draw

`CString blabla;` //local string

`blabla.Format( "Umfang=%d, Flaeche=%d", int(perimeter), int(area) );` //Convert `perimeter` and `area` to integers and put them into the string.

`dc.TextOut( 0, 140, blabla );` //Display the string about 7 lines below the upper border.

`dc.Rectangle( minmax );` //Draw a white Rectangle with black borders.

`dc.Polygon ( polygon, n );` //Draw a closed polygone onto the rectangle.

`dc.Rectangle( center_of_gravity.x-3, center_of_gravity.y-3, center_of_gravity.x+3, center_of_gravity.y+3 );` //Draw the center of gravity as small rectangle onto the polygone.

In Version 6 `CChildView.cpp` inside `CChildView :: OnMouseMove(...)`

`dc.Polyline( polygon, n );` //Draw an open polyline from vertex 0 to n-1.

`switch ( n%3 )` //Divide n by 3 and retain the remainder. The Windows API accepts only Bezier curves having the lengths 4, 7, 10, 13 ...

`case 0: dc.PolyBezier( polygon, n-2 ); break;` //Shorten the polygone by two vertices and draw it

`case 1: dc.PolyBezier( polygon, n ); break;` //Draw the polygone

`case 2: dc.PolyBezier( polygon, n-1 ); break;` //Shorten the polygone by one vertex and draw it
