Courses 2DCx, C3: Animation, FAQs

Copyright © by V. Miszalok, last update: 20-01-2001

Was ist interessant an der Funktion CChildView::OnCreate(...) ?

Diese Funktion reagiert auf die Nachricht des Betriebssystems `WM_CREATE`.
`WM_CREATE` wird ausgeschickt, wenn ein Fenster fertig aufgebaut und funktionsfähig ist. Dies ist ein guter Zeitpunkt um Initialisierungen vorzunehmen, die am Anfang und nur einmal in der Lebenszeit des Fensters fällig sind. Im Musterprogramm anim1 sind das die Initialisierung der Variablen `fertig, zoom, bogen, sinus, cosinus` und das Starten eines Timers. Man könnte das Initialisieren natürlich auch im Konstruktor `CChildView()` machen, aber wir werden `CChildView::OnCreate(...)` im nächsten übungsprogramm `FirstOpenGL` dringend brauchen und führen es deshalb jetzt schon ein.
Die Initialisierung von `zoom = 0.995f;` betrifft nur den allerersten Schrumpfungsvorgang der Animation, dann wird `zoom` zyklisch neu gesetzt am Ende der `OnTimer(...)`--Funktion.
Wichtiger ist das Setzen von `double bogen = 3.14159 / 180.;`. Diese Zeile setzt das Rotieren auf Schrittweiten von ein Grad im Uhrzeigersinn. Hier liegt ein wichtiger Hebel zur Bremsung, Beschleunigung und Umkehrung der Rotation.

Was ist ein "Timer" ?

`SetTimer( 1, 1, NULL );` ist eine Memberfunktion der `CWnd`-Klasse (von dieser ist `CChieldView` abgeleitet).
Parameter Nr. 1 ist die Nummer des Timers (man kann mehrere gleichzeitig in Gang setzen).
Parameter Nr. 2 ist die Anzahl der Millisekunden.
Parameter Nr. 3 wird hier nicht benutzt.
Das Betriebssystem schickt nun pausenlos `WM_TIMER`-Nachrichten an unser `CChieldView`-Fenster, das `SetTimer();` angefordert hat.
Diese Nachrichten stapeln sich nicht, sondern die jeweils letzte tötet alle Vorgängerinnen. Das Betriebssystem ist in Wirklichkeit nicht in der Lage, exakt jede Millisekunde eine solche Nachricht zu verschicken. Es tut es nur dann, wenn es nichts wichtigeres zu tun gibt. Die eine Millisekunde bedeutet eigentlich: "Schicke die Nachrichten so schnell du kannst". Ist man auf eine einigermassen exakten Zeitrythmus angewiesen, dann muss man den 2. Parameter auf etwa 50 Millisekunden erhöhen.
Die `WM_TIMER`-Nachrichten machen nur Sinn, wenn man eine `CChildView::OnTimer(...)`-Funktion programmiert, die auf `WM_TIMER` reagiert (siehe unten).
Dieser Mechanismus aus `SetTimer(...);` und `OnTimer(...)`-Funktionen ist das wichtigste Uhrwerk jeder Animation. `SetTimer(...);` wirkt als Metronom und Taktstock der Animation wie eine Nockenwelle in einem Verbrennungsmotor. Parallele Timer sind möglich, so daß ein Fenster mehrere unabhängige Animationen haben kann. Dann muss innerhalb von `OnTimer(nIDEvent)` eine Fallunterscheidung an Hand des Parameter `nIDEvent` programmiert werden.

Warum werden die Polygonkoordinaten p[i] in FLOAT-Koordinaten f[i] umgerechnet ?

Die Message-Handler-Funktion `OnMouseMove(nFlags, point)` liefert INTEGER-Koordianten, nämlich Spalte und Zeile der Rastermatrix. Wir merken uns diese Koordinaten in Form des Arrays `CPoint p[nMax];` und zeichnen das Polygon mit `dc.MoveTo( ix, iy );` und `dc.LineTo( ix, iy );`, die INTEGER-Koordinaten erwarten.

Das Musterprogramm führt parallel zu `p[nMax]` einen gleichdimensionierten Array `FPoint f[nMax]`, dessen Elemente wir als FLOAT-Koordinaten deklariert haben. `f[nMax]` wird gefüllt am Ende von `OnLButtonUp(...)` durch die Schleife:

```
for ( i = 0; i < n; i++ )
{   f[i].x = float(p[i].x - m.x); //m = INTEGER-Mittelpunkt des
    f[i].y = float(p[i].y - m.y); //umschreibenden Rechtecks
}
```

Begründung 1:
Der Zoomfaktor `zoom` und die Drehfaktoren `sinus` und `cosinus` können unmöglich als INTEGER deklariert werden. Deren Werte sind immer in der Nähe von 1.0 bzw. zwischen -1.0 und +1.0. Multiplikationen mit solchen Werten kann man nur in FLOAT (oder DOUBLE) programmieren, auch dann wenn INTEGER in die Formel eingehen und herauskommen sollen. Umwandlungen von INTEGER in FLOAT brauchen Zeit. Es ist extrem unwirtschaftlich, sie in `OnTimer(...)` auszuführen, wir legen sie deshalb nach `OnLButtonUp`
Begründung 2:
Unsere Animation kaskadiert in dem Sinn, dass das FLOAT-Polygon `f[nMax]` um 5 Promille vergrößert und um ein Grad gedreht wird. Der FLOAT-Output einer Kaskadenstufe ist der FLOAT-Input der nächsten Kaskadenstufe, ohne dass das INTEGER-Polygon `p[nMax]` noch irgend eine Rolle spielt. Am Ende jeder Kaskadenstufe runden wir die FLOAT-Koordinaten nach INTEGER, um sie zu zeichen, aber nicht die gerundeten Koordinaten gehen ein in die nächste Kaskadenstufe, sondern die ungerundeten FLOAT-Koordinaten. Andernfalls würden sich die Rundungsfehler unkontrolliert fortpflanzen, nach wenigen Stufen würde der Maler sein Polygon nicht mehr wiedererkennen.

Warum verschiebt man den Mittelpunkt des Polygons auf den Nullpunkt und wieder zurück ?

Im der for-Schleife des vorigen Absatzes wird das INTEGER-Polygon `p[nMax]` nicht nur in ein FLOAT-Polygon `f[nMax]` umgewandelt, sondern nebenbei auch noch gescrollt und zwar derart, dass der Polygonmittelpunkt in die linke obere Fensterecke (=Koordinatenursprung) verschoben wird. Beim Zeichnen nach dem verschieben würde man nur noch das rechte untere Viertel des Polygons sehen, weil der Rest außerhalb des Fensters unsichtbar bleiben würde. Nach Zoom und Rotation wird das Polygon an den alten Platz zurückverschoben bevor es sich in das Fenster zeichnet.
Begründung:
Das Polygon wird von Vektoren aufgespannt, die alle im Koordinatenursprung fußen und an den Polygonecken ihre Spitze haben. Zoom verlänger/verkürzt die Vektorenspitzen (Betragsänderung) und die Rotation dreht um den Fußpunkt. Das bedeutet, dass alle Streckungen ihr Streckungszentrum und alle Drehungen ihr Drehzentrum implizit im Nullpunkt haben.
Dies entspricht nicht den Erwartungen an Zoom und Rotation. Man erwartet einen Zoom und eine Rotation um ein Zentrum im Inneren des Polygons, am besten in der Mitte des Polygons. Als Mitte kommt gefühlsmäßig der Eckenschwerpunkt in Frage oder der Flächenschwerpunkt oder die Mitte des umschreibenden Kreises oder die Mitte des umschreibenden Rechtecks. Letzere ist leicht zu berechnen und wird deshalb im Musterprogramm benutzt.
Der User und Betrachter des Programms weiß nichts von der Existenz und der Lage eines Koordinatensystems. Er erwartet einen Zoom und eine Rotation "am Ort", d.h. ohne Platzwechsel des Polygons.
Mathematisch erfordert das zwei (dem User/Betrachter unsichtbare) Koordinatentransformationen, nämlich Verschiebung des Streck- und Drehpunkes auf den Koordinatenursprung und Rückverschiebung auf den alten Platz, nachdem Zoom und Rotation berechnet wurden.
Machen Sie das Experiment und unterlassen sie diese Verschiebungen, indem Sie schlicht `m.x = m.y = 0;` setzen. Sie werden sehen, das Polygon wandert weitgehend unsichtbar irgendwo außerhalb des Fensters während es sich um die linke obere Fensterecke streckt und dreht.

Warum MoveTo+LineTo statt Polyline(...) ?

In der `OnTimer(...)`-Funktion steht der Befehl:
`if ( !i ) dc.MoveTo( ix, iy ); else dc.LineTo( ix, iy );`
Im Klartext: Zum ersten Punkt des Polygons keine Linie ziehen, nur hinspringen. Zu allen weiteren Punkten Linien ziehen.
Es gibt eine viel elegantere Zeichenmethode für ein Polygon: `dc.Polyline( p, n);`
Frage: Warum wird die elegante Methode nicht benutzt ?
Begründung:
`dc.Polyline( p, n);` setzt voraus, dass die Polygonpunkte in einem Array `CPoint p[nMax]` gespeichert sind. Aus schon genannten Gründen sind die Polygonecken in einem FLOAT-Array `FPoint f[nMax]` gespeichert. So einen Datentyp akzeptiert die Methode `Polyline(...)` nicht. Um sie zu benutzen, müsste man f[i] zunächst in p[i] umwandeln. Das lohnt nicht, `dc.LineTo(...), dc.MoveTo(...)` akzeptieren einzeln Punkte ohne Arrayform.

Was bedeutet Invalidate() ?

Dies ist eine Memberfunktion der Klasse `CWnd`, von der `CChildView` abgeleitet ist. Sie teilt dem Betriebssystem mit, dass der alte Fensterinhalt zerstört, ungültig oder zumindest unwichtig geworden ist.
Das Betriebssystem reagiert mit Löschen und schickt an `CChildView` eine WM_PAINT - Message um mitzuteilen, dass gelöscht ist und ein komplettes Neuzeichnen des Fensterinhalts erwartet wird.
Bei schnellen Animationen muss man oft löschen und das Polygon an etwas anderer Stelle neu zeichnen. Die Folge ist ein unangenehmes Flickern, das schwer zu bekämpfen ist.
In diesem Fall hätte `Invalidate( false )` die Wirkung, daß das Betriebssystem das Fenster nicht löscht. Sonst bleibt der Mechanismus identisch. Das kann sinnvoll sein bei Animationen, die nicht unbedingt auf Löschen angewiesen sind. Machen Sie das Experiment und benutzen Sie zur Probe `Invalidate( false )` anstatt `Invalidate()`.