Courses 2DCx: 2D-Computer Graphics with C++/MFC
Chapter C1: Frequently Asked Questions to the Intro Project

(teilweise in Anlehnung an David Kruglinski: Programming MS Visual C++, Fifth Edition, Microsoft Press 1998 pp. 1-16)

Was ist die beste Sprache für Computer Graphics, Image Processing und Computer Vision ?

Teilen wir die Sprachen in
Typ 1: hardwarenahe mit Compiler: z.B. Fortran, Pascal, C++ (Vorteil: schnelle Ausführung, Nachteil: nur auf einer Plattform lauffähig)
Typ 2: hardwareferne ohne Compiler: z.B. Java, VisualBasic, C#, JScript (Vorteil: auf diversen Plattformen lauffähig, Nachteil: langsame Ausführung).
Für Image Processing und Computer Vision ist die Abwägung meistens klar: Redundanz und Datenmenge verlangen Typ 1. Es gibt einfach keine Obergrenze des Datenvolumens, jeder technische Fortschritt (CPU, Speicher, GraphicChips, Bandbreite) ist immer nur ein Tropfen ins Feuer der explodierenden Redundanz der multimedialen Datenströme.
Für Graphik mit hoher Ortsauflösung (Bilder > 10 MByte) oder Zeitauflösung (Action-Games) oder beidem ist C++ unschlagbar schnell.

Für Computer Graphics ist Typ 2 aber in folgenden Fällen interessant:
1. Für Mensch-Maschinen-Schnittstellen (GUIs=GraphicalUserInterfaces), wo menschliche Eingaben (Tastatur/Maus) das Tempo bestimmen.
2. Für kleine Animationen im GUI und im WEB.
3. Wenn DirectX direkt aus der Sprache zugänglich ist, wie bei XNA und WPF.

Was ist Windows ?

W. ist eine Familie von graphischen Betriebssystemen (3.1, 95, 98, Millenium, NTWorkstation, NTServer, 2000Professionell, 2000Server, XP, Vista, CE), die (wie MacOS, Linux, OS/2) auf die grundlegenden Entwicklungen von Xerox und Apple aus den 70er Jahren aufbauen. Man spricht von Familie und Generationen, weil bei Microsoft das jeweils neueste Betriebssystem rückwärtskompatibel zu seinem Vorgänger ist.
Vorteil der Rückwärtskompatibilität: Der User kann ohne Verluste an Applikationen auf ein neues Betriebssystem updaten.
Nachteil: Im Inneren des jeweils neuesten Betriebssystems müssen enorme Mengen von alten Konzepten und altem Code mitgeschleppt werden, damit die alten Applikationen weiter laufen. Die Entwickler versinken langsam im Sumpf ihrer alten Fehler, wenn sie die Brücken nach rückwärts nicht abbrechen dürfen.

Was war neu an den graphischen Betriebssystemen von Xerox und Apple ?

Früher (z.B. bei DOS): Das Anwendungsprogramm ruft das Betriebssystem, wenn es User Input braucht. Das Anwendungsprogramm kommandiert damit das Betriebssystem und den User.
Neu: (bei Windows, MacOS, Linux etc.): Das Betriebssystem ruft das Anwendungsprogramm, wenn User Input da ist. Das Betriebssystem und der User kommandieren damit das Anwendungsprogramm.
Diese Umkehrung der Kommandorechte erzwingt eine neue Art von Anwendungsprogrammierung, die sogenannte „ereignisorientierte“ Programmierung.
Ereignisse sind Signale aus der Hardware (Tastatur, Maus, Graphikkarte, Harddiskcontroller, Netzwerkkarte etc.), die das Betriebssystem selbstständig bearbeitet. Anwendungsprogramme bekommen Kopien dieser Ereignisse (Nachrichten=WindowsMessages) und können die Bearbeitung auf Antrag zu sich selbst umlenken.
Die Autonomie des modernen Programmierers ist viel kleiner als der Anfänger vermutet. Das moderne Betriebssystem lässt wenig Eigenmächtigkeiten zu. Kurz: Windows-Programmierung bedeutet Surfing auf dem Betriebssystem. Wie beim richtigen Surfing steht der Anfänger mehr im Wasser als auf dem Brett.

Was ist das Windows API ?

Zum Zwecke des Surfens sind von den ca. 1 Million Funktionen des Betriebssystems ca. 3000 offengelegt, d.h. können von jedem Programmierer (etwa von Pascal, Delphi, C und Basic) aus aufgerufen werden. Diese offengelegten Funktionen heißen kollektiv das Application Programming Interface (API).
Beispiele für zwei wichtige API-Funktionen:

```
hwnd = CreateWindow( .... );
ShowWindow( hwnd );
```

Das bedeutet: Reserviere Platz für eine Datenstruktur, die ein Fenster beschreibt und fülle sie mit vernünftigen Werten und zeige das Fenster am Bildschirm.
Man muss sich vor Augen halten, welche ungeheure Komplexität sich hinter diesen zwei API-Aufrufen verbirgt: Auf so gut wie jedem Monitor dieser Welt erscheint jetzt ein bewegliches Fenster mit Titel, Rahmen; Farben, Reaktionen etc.
Ein großes Problem ist, dass der Programmierer kaum Einfluss darauf hat, wann sein Fenster erscheint, ob es mit anderen Fenstern harmoniert und ob Tastatur und Maus mit seinem Fenster zusammenarbeiten. Der Programmierer kann zwar entsprechende Anträge an das Betriebssystem stellen, aber er kann nie sicher sein, ob und wann das Betriebssystem den Anträgen stattgibt. Die einzelnen Prozesse des Betriebssystems und die Anwendungsprogramme synchronisieren sich, indem sie sich gegenseitig Nachrichten (Windows Messages) senden. Es gibt ca. 100 solcher Nachrichten. Beispiele WM_CREATE ist die Nachricht, dass ein Fenster aufgebaut wird, WM_LBUTTONDOWN ist die Nachricht, dass der User die linke Maustaste gedrückt hat, WM_CHAR ist die Nachricht, dass der User einen Buchstaben der Tastatur gedrückt hat, WM_COMMAND ist die Nachricht, dass der User ein Menülement oder einen Dialogbutton angeklickt hat, WM_CLOSE ist die Nachricht, dass der User oder das Betriebssystem ein Fenster vernichten will,. Das Schreiben von Programmen bedeutet im wesentlichen das Schreiben von Funktionen, die auf Nachrichten reagieren und die ihrerseits Nachrichten versenden. (Mit MFC braucht man nicht einmal mehr ein Hauptprogramm zu schreiben, WinMain bleibt irgendwo versteckt.)
Man musste früher (bis ca. 1996) ziemlich viele API-Funktionen und deren Zusammenspiel kennen, um nützliche Programme schreiben zu können und schlimmer, man musste immer wieder die gleichen Ketten von API-Funkionen hintereinander schreiben, für Aufgaben, die sich wiederholten.

Was sind die MFC 9.0 ?

Die **Microsoft Foundation Classes** bieten einen leichteren und moderneren Zugang zu Windows als das API.
Die Grundidee ist: Man fasse für möglichst viele Lebenslagen des Programmierens zusammengehörige API-Funktionen zu Klassen zusammen. Oder anders ausgedrückt, man reduziere die ca. 3000 öffentlich zugänglichen API-Funktionen auf ca. 200 Klassen. Man verpacke also in jede Klasse soviel API-Funktionen wie irgend möglich und sinnvoll, weil es leichter ist, ca. 200 Klassen zu überblicken als 3000 einzelne API-Funktionen.
Die Randbedingungen waren:
Es soll nach wie vor erlaubt sein, jederzeit in- und außerhalb der MFC jede API-Funktion direkt aufzurufen. API-Aufrufe werden nicht von den MFC monopolisiert sondern sind nach wie vor frei zugänglich.
Ob man eine API-Funktion direkt oder in einer MF-Klasse als Methode benutzt, darf keinen Laufzeitunterschied machen. Klassenmethoden dürfen nicht langsamer sein, als die entsprechenden API-Funktionen.
Die Klassen dürfen keinen zusätzlichen Speicherplatz beanspruchen, den das Betriebssystem nicht sowieso benötigt. Es dürfen also keine Daten doppelt geführt werden in der Klasse und im Betriebssystem, sondern die Klassen müssen direkt in die Betriebssystemdaten lesen und schreiben.
Die jetzt zu Verfügung stehende MFC-Bibliothek ist populär geworden, weil sie diese Anforderungen mehr oder

weniger erfüllt. Sie schafft Ordnung im API-Chaos, sie bietet ein objektorientiertes Programmiergerüst für fast alle Lebenslagen, sie kanalisiert die Freiheit des Programmierers ohne ihn zu etwas zu zwingen und sie verbraucht keine nennenswerten zusätzlichen Ressourcen (Laufzeit und Platz).

Was ist VisualStudio ?

Visual Studio ist eine integrierte Entwicklerumgebung (Integrated Developer Environment = IDE), oberhalb von VisualC++, das eine Arbeitsoberfläche und gemeinsame Editoren bietet für VisualC++, Visual Basic, Visual J++, das HTML-Hilfetextsystem MSDN = Microsoft Developer Network und Visual Source Safe (ein Verriegelungssystem, das Chaos vermeidet, wenn mehrere Leute an einem einzigen Projekt arbeiten). Mit Visual Studio kann man zwischen den oben genannten Sprachen und Werkzeugen umschalten, ohne die Oberfläche wechseln zu müssen, was ziemlich praktisch ist.

Sein Schwerpunkt liegt in Werkzeugen zur Entwicklung von Programmen, die als Client und Server im WEB (ohne Festlegung auf irgendeine Hardware) verteilt sind. Diese Programme kommunizieren über XML-Dokumente und ein offenes Simple Object Access Protocol = SOAP.

Das neue Visual Studio von Microsoft bietet eine erstaunliche Vielfalt von Typ1- und Typ2-Sprachen parallel unter einem Dach (siehe erstes Kapitel dieser Seite)
Typ 1: C++ pur (=Win32Application) oder C++ mit MFC.
Typ 2: VisualBasic, C#, JScript, Managed C++, XAML.
Microsoft offeriert in Visual Studio zusätzliche interessante Kreuzungen von Typ2- mit Typ1-Sprachen, nämlich die Möglichkeit, Typ2-Code vor der ersten Ausführung dauerhaft in lokal im Zielrechner residenten Typ1-adäquaten Maschinencode umzuwandeln (dann ist nur noch das erste Mal alles langsam).

Was ist VisualC++ ?

VisualC++ enthält zwei getrennte Entwicklungssysteme für C-Programme in einem Produkt.
1. Man kann mit C++ und der Win32 API Windowsprogramme schreiben, ohne jede Objektorientierung und ganz ohne Klassen (Diese Methode beschreibt Charles Petzold in seinem klassischen Buch: Programming Windows 95, Microsoft Press 1996.)
2. Man kann mit C++ und dem MFC-Application Framework (Anwendungsgerüst beschrieben in MS VisualC++ MFC Library Reference, Microsoft Press 1997) objektorientiert Windowsprogramme schreiben.
Beide Methoden schließen sich gegenseitig nicht aus. Man kann sie beliebig mischen und es gibt viele Fälle, in denen das mischen sinnvoll ist.

Was kann der Source Code Editor ?

VisualC++ enthält einen sehr guten makrofähigen Codeeditor mit Syntax-Färbung, automatischer Einrückung und AutoComplete. Sie brauchen nur den Anfang eines Befehls zu tippen und der Editor macht Vorschläge für den Rest des Befehls. Das ist ideal für OOP, tippen Sie den Klassennamen und der Editor listet automatisch alle Membervariablen und Methoden zur Auswahl. Ähnlich komfortabel listet er die Parameter fast aller Win32 API-Funktionen.

Was können die Resourcen-Editoren?

Resourcen sind Menus, Dialoge, Error-Texte, Bilder etc., die getrennt vom Programmcode erzeugt und in eigenen Files (Resourcen-Files) abgelegt werden. Für sie gibt es eigene Resourcen-Compiler und deren binärer Output wird mit dem binären Output des C++-Compilers im letzten Lauf des Linkers zu einem *.EXE oder *.DLL vereinigt. Wenn Sie auf die Resourcenansicht klicken, dann können Sie Resourcen für Ihr Programm zum editieren auswählen. Es öffnet sich jeweils ein Resourceneditor, der zur geöffneten Resource passt, etwa ein Wysiwyg-Editor, wenn Sie Menus editieren, ein Graphikeditor, wenn Sie Dialoge editieren, ein Pixeleditor, wenn Sie Icons editieren usw. Jedes Projekt hat ein Textformat-Resourcen-Script-File *.RC, welches die Menus, Dialoge, Error-Texte, Bilder und Pfade zu externen Include-Dateien beschreibt. Sie bekommen dort einen Überblick, welche Resourcen in Ihr Programm eingebunden sind. Es ist allerdings nicht zu empfehlen, dieses RC-File von Hand zu verändern. Wenn Sie dabei einen Fehler machen, dann streiken alle Resourcencompiler und es kommt kein *.EXE mehr zustande. Interessant ist, dass man fremde *.EXE und *.DLL mit den Resourceneditoren öffnen kann und die Resourcen der fremden Programme frei editierbar vor sich sieht.

Was ist ein Projekt ?

Ein Projekt ist eine Sammlung von zusammenhängenden Quelldateien, die gemeinsam compiliert und gelinkt werden, um ein EXE- oder DLL-File zu erzeugen. Projekt-Files (extension *.DSP) sind die Nachfolger der früheren Make-Files (extension *.MAK). Sie haben Textformat und sind frei editierbar. Mehrere Projekte können in einem Workspace vereinigt sein, deshalb gibt es noch Workspace-files (ebenfalls Textformat, Extension *.DSW).
Visual Studio erstellt neue *:DSP und *.DSW automatisch, wenn immer Sie anfangen zu programmieren. Wollen Sie vorhandene Projekte öffnen, doppelklicken Sie auf das zugehörige *.DSW und Visual Studio startet und restauriert Projekt und Workspace wie sie ihn verlassen haben.
File Extensions der Projektdateien

```
DSW   Workspace file
DSP   Project file
MAK   External makefile*
DEP   Dependency file*
OPT   Holds workspace configuration*
APS   Supports ResourceView*
BSC   Browser information file*
CLW   Supports ClassWizard*
NCB   Supports ClassView*
PLG   Builds log file*
```

*Diese Files kann man ohne Probleme löschen. Sie werden beim nächsten Übersetzen automatisch neu erzeugt.
Ein Projekt benötigt zum übersetzen und linken nicht nur die oben erwähnten zusammenhängenden Quelldateien sondern meistens noch viele Files von außerhalb des Projekts (externe include files, library files etc).
Visual Studio ersetzt die Projekte durch ein übergeordnete Datenstruktur: "Solution" (File name extension = `*.sln`), die man automatisch aus den VC++ `*.prj`-Files erzeugen kann.

Was macht der Anwendungsassistent (AppWizard) ?

Der AppWizard ist ein Code-Generator, der am ersten Anfang eines neuen Projekts eine `*.DSW` und eine `*.DSP` Datei sowie ein Skelett=Anwendungsgerüst=Application Framework einer Windowsapplikation erstellt mit allen Klassen, Files und Zubehör, das Sie auswählen. Vorteil: Man kann sehr schnell Programme schreiben. Nachteil: Der vom AppWizard erzeugte Skelett-Code ist so allgemein, dass er unleserlich und unverständlich bleibt. Man hat sogar den Eindruck, jemand hat ihn künstlich verkompliziert und unverständlich aufgebläht. Andererseits sind die Macros veröffentlicht, die den Code erstellen und Microsoft ermöglicht damit jedermann, eigene AppWizards zu schreiben.
Vorschlag: Nutzen Sie den Komfort des Microsoft-AppWizard und akzeptieren Sie ohne zu fragen das automatisch erzeugte Codeskelett. Es hat sich schlicht bewährt.

Was macht der Klassenassistent ?

Der Klassenassistent ist ebenfalls ein Codegenerator, der in ein vorhandenes Projekt neue Klassen, Membervariable, virtuelle Funktionen, MessageHandlerFunktionen einbaut. Der ClassWizard schreibt die Prototypen, die Funktionsköpfe und den Code, der Windowsnachrichten mit Ihren Funktionen verbindet. Er vergisst nichts und ordnet alles in alphabetischer Reihenfolge (letzteres kann auf die Nerven gehen). Sie starten den Klassenassistenten aus dem Ansicht-(=View) Menü des Compilers.

Wie funktioniert Online Help ?

VisualC++ besitzt kein eigenes Hilfesystem. Dieses ist ausgelagert in eine eigene Applikation, die unabhängig vom (aber nach dem) Compiler installiert werden muss. Diese Applikation heißt Microsoft Developer Network (MSDN-) Library Viewer. Die gesamten Hilfetexte der MSDN Library sind in HTML geschrieben. Der Viewer ist nichts weiter als eine Untermenge des Internetexplorers und arbeitet deshalb wie ein normaler Webbrowser. Die HTML-Help-Texte können physikalisch von einer Harddisk, von CD oder Online aus dem Internet kommen. Der Programmierer kann die ungeheure Menge an Information nach Büchern (by book) oder nach Themen (by topic) oder nach Stichwörtern(by word) ordnen oder kann sie schlicht mit der F1-Taste aus dem Code direkt aufrufen. MSDN ist die zentrale Dokumentation für alle Microsoftcompiler, Interpreter und Officeapplikationen. Es enthält derzeit etwa 1 Mio HTM-Dokumente. Bei schneller Internetverbindung braucht man MSDN nicht lokal auf der Festplatte. Die F1-Taste holt die Dokumente auch direkt von den (sehr schnellen) Microsoft-Servern.