

VGA Graphik Teil 2

Projektplanung und Programmwurf

So, da wir nun im Teil 1 uns einen groben Überblick über die Hardware einer VGA Karte verschafft haben, und die Spezifikationen des VGA Standards kennen gelernt haben, machen wir uns im Teil 2 daran die Anforderungen an ein Programm mit/zur graphischen Darstellung festzustellen. Solch eine recht systematische Vorgehensweise hat den Vorteil das wir exakter vorhersagen können ob unsere Anwendung auf dem beabsichtigten Zielsystem läuft, oder ob ggf. Probleme auftreten könnten, und was zu tun ist damit diese gelöst werden können.

Es gibt allerdings auch viele Fälle wo eine umfangreiche Projektplanung etc. gar nicht erforderlich ist, Beispiel: um mir einen Überblick über eine Tendenz zu verschaffen gebe ich die Werte einer math. Funktion graphisch aus, eine aufwendige 'Projektplanung' entfällt, da es praktisch ein 'Einwegprogramm' ist.

Wenn wir aber Programme schreiben möchten, die auch von anderen benutzt werden sollen, und auch evtl. auf anderen Zielsystemen als dem Entwicklungssystem laufen, dann kommen wir um die Planung und einige ganz grundsätzliche Überlegungen nicht drumherum.

Bevor wir anfangen jedoch eine Überlegung ganz banaler Art: reicht unserer Anwendung die Auflösung/Farbtiefe des VGA Modus? Bei einer Reihe statistischer/mathematischer/technischer Anwendungen und vielen Spielen kann man sicherlich ein klares 'Ja' formulieren, sollte die Antwort allerdings auf ein 'Nein' hinauslaufen, müssen wir uns Gedanken machen höhere Auflösungen zu benutzen, ob man nun SVGA / XGA Modi benutzt oder lieber gleich eine OpenGL Anwendung verfasst muss man dann im Einzelfall entscheiden.

Als erstes wollen wir sicherstellen das nach dem Beenden des Graphikprogramms wieder alles ist wie vorher, also der Videomodus vor dem Programmstart ohne Einschränkungen nutzbar ist. Notieren wir einfach mal die Vorgehensweise:

1. Feststellen und Abspeichern des aktuellen Videomodus
2. Eingaben von Werten im Textmodus (falls erforderlich)
3. Aktivierung des Graphikmodus
4. Sichern der alten Palette und der Registerinhalte (falls erforderlich)
5. (Laden der eigenen Palette / Einstellung der VGA Register)
6. Hauptprogramm / Bilddarstellung
7. (Wiederherstellung der alten Palette / Registerinhalte)
8. Wiederherstellung des alten Videomodus
9. Programmende

Kurz gesagt: wenn wir uns an dieses Schema halten, sollte nach dem Beenden des Graphikprogramms der vorherige Videomodus störungsfrei nutzbar sein. Jetzt folgen noch die wichtigsten Erwägungen die es bei Erstellung eines Graphikprogramms zu beachten gilt.

Auflösungen, Farbanzahl und Videomodus

Sind wir einmal ehrlich: wie oft wird im Bereich Computergraphik hemmungslos mit 'Kanonen auf Spatzen' geschossen? Unnötig hohe Auflösungen verbessern z. B. ein Balkendiagramm höchstens marginal, fressen aber Speicherplatz und Ausführungsgeschwindigkeit. Überlegen wir uns also welcher Videomodus für welche Anwendung angemessen sein mag:

Textmodus/Modus 03h: Durch einige vorgefertigte Graphikzeichen (siehe Zeichensatztabellen) reicht dieser absolut für einfache Balken- oder auch Blockdiagramme.

EGA Modus 013h / 320 x 200 Pixel 256 Farben: der Modus ist ideal für Spiele, Demos und evtl. einfache Simulationen, die Auflösung ist nicht gerade berauschend, aber dieser Modus ist dafür recht einfach in der Programmierung.

VGA Modus 012h / 640 x 480 Pixel 16/256 Farben: graphische Darstellung von Funktionen, Messwerten oder ähnlichem dürfte hier schon gut möglich sein, bedingt schon das Darstellen von Bildern, wie z.B. Vorschau von BMP Dateien

SVGA/XGA Modi/VESA: für Bildbearbeitung/Bildmanipulation reichen die oben beschriebenen Auflösungen keinesfalls, hier sind größere Auflösungen erforderlich, außerdem ist es beispielsweise sinnvoll OpenGL oder DirectX Modulbibliotheken zum Laden, Umwandeln und Speichern der verschiedenen Bildformate zu benutzen.

Verarbeitungsgeschwindigkeit

Spielt die Verarbeitungsgeschwindigkeit eine Rolle muss man in jedem Fall verschiedene Aspekte berücksichtigen: eine unnötig hohe Auflösung und/oder Farbanzahl steht einer hohen Verarbeitungsgeschwindigkeit im Weg. Unter Umständen sind hier nur Kompromisse möglich. Höchste Verarbeitungsgeschwindigkeiten sind mit Hochsprachen kaum zu realisieren, und auch mittels Assembler sind nicht alle Methoden förderlich, beispielsweise beansprucht die Verarbeitung von Fließkommazahlen mittels des Coprozessor relativ viel Zeit, sodass ein flüssiger Bildaufbau nicht mehr gewährleistet ist. Klassifizieren wir ruhig einmal die verschiedenen Verarbeitungsgeschwindigkeiten nach Datenquellen um einen Überblick zu erhalten:

1. Handeingabe, diese ist in der Regel nicht Zeitkritisch
2. Erfassung kleinerer bis mittlerer Datenströme, nehmen wir als Beispiel das Aufzeichnen von Temperaturwerten, die Daten kommen kontinuierlich, aber dennoch in relativ langsamen Zeitintervallen, das sollte auch noch von Interpreter Sprachen wie BASIC, Python und anderen zu handeln sein.
3. Erfassung mittlerer bis großer Datenströme, nehmen wir einen Elektromotor auf dem Prüfstand und sagen wir es wird Drehzahl, Stromaufnahme, Drehmoment und anderes bei wechselnden Lastzuständen erfasst, der Bediener betrachtet das Diagramm und muss unter Umständen den Versuch abbrechen können, dieses wäre ein Fall klassischer Echtzeitdatenverarbeitung.
4. Videostream: diese Datenströme lassen sich in der Regel zwar echtzeitig erfassen, beispielsweise zur unmittelbaren Darstellung, aber zur Verarbeitung wie z.B. der Anwendung von Videofiltern, führt kein Weg am Zwischenspeichern vorbei

Für das praktische Programmdesign reicht erstmal diese grobe Unterteilung, die Problemstellungen im einzelnen erfordern mitunter sehr kreative Lösungen.

Die zur Verarbeitung zur Verfügung stehende Zeit bedingt auch mittels welcher Methode man das Bild erzeugt, bei langsamen Vorgängen wären die Video BIOS Interrupts nutzbar, soll die Anwendung echtzeitfähig sein, dann ist der direkte Zugriff auf den Videospeicher wohl eher die erste Wahl.

Speicherplatzbedarf

Das ist natürlich auch ein Entwicklungskriterium. Vor dem Laden oder Erstellen entsprechender Graphiken muss ausreichend Speicher reserviert sein.

Unkomprimierte Rastergraphiken (Bitmaps) benötigen am meisten Speicherplatz, während Vektorgraphiken relativ sparsam sind. Und so ganz am Rand (Praxistip): der Stapelspeicher sollte bei einer Graphikanwendung wenigstens ~200Bytes umfassen, ist er zu klein bricht die Anwendung unter Umständen ohne Fehlermeldung zusammen.

Kompatibilität

Ist die spätere Laufzeitumgebung das Entwicklungssystem dann braucht man hier nichts zu berücksichtigen. Ist das Zielsystem aber ein anderes, evtl. mit einer anderen Graphikkarte bestückt, könnte es problematisch werden. Ein exakt für den VGA Standard entwickeltes Programm sollte in der Regel auch auf einem Zielsystem nach VGA Standard lauffähig sein, gleiches gilt so für Programme die die VESA Standards berücksichtigen. Alle anderen 'Standards' (SVGA, XGA usw.) sind lange nicht so scharf umrissen und somit kann man keine Kompatibilität garantieren, hier ist der Ausweg mittels OpenGL zu arbeiten und somit hardwareunabhängig zu sein.

Zusammenfassung

Ein Programm mit graphischer Ausgabe sollte die VGA Karte in dem Zustand an das Betriebssystem zurückgeben wie es sie vorgefunden hat. Bei vielen Programmen spielt die Verarbeitungsgeschwindigkeit eine große Rolle, deshalb sind Compilersprachen oder Assembler den Interpreter Sprachen vorzuziehen, manche Anwendungen lassen sich auch heute nur in Assembler realisieren. Weiterhin ist die Auflösung und Farbanzahl relevant: 'nur' 1000 x 1000 Pixel mit 24 Bit Farbtiefe bedeutet auch das das Programm drei Millionen Byte pro Einzelbild berechnen/transportieren muss, also sollten die zum Bildaufbau/Bildberechnung benötigten Algorithmen kurz und effektiv sein. Oftmals entscheidet das Programmdesign nicht über „Funktion / nicht Funktion“, sondern über „gute Bilddarstellung / Qualität“ oder „schlechte Bilddarstellung / Qualität“.

Schliessen wir hier Teil 2 mit einer 'goldenen Regel':

In die Programmschleife des Bildaufbaus kommt nur das was auch zum Bildaufbau nötig ist, alle anderen Werte, die sich zur Laufzeit der Graphikdarstellung nicht ändern, werden vorher berechnet.