Stolpersteine und Spezialitäten in C/C++
betreut von Aegidius Plüss, Gymnasium Bern-Neufeld

Der Computer als Messgerät – ein Thema, so alt wie die Computer selbst. Immer schon und immer wieder wird der Computer dafür eingesetzt, Messdaten aufzunehmen, zu verarbeiten und Systeme zu steuern. Man schätzt, dass weit über die Hälfte der weltweit eingesetzten Computerleistung in diesem Zusammenhang Verwendung findet. Dies beginnt beim Microcontroller im Küchengerät, geht über Prozessrechner im modernen Auto, hin bis zum Hochleistungsrechner zur Steuerung von Teilchenbeschleunigern. Aber auch im naturwissenschaftlichen Unterricht hält der Computer seit mehreren Jahren Einzug als Messgerät. Unzählige Artikel und Kurse behandeln das Thema „Messen, Steuern, Regeln - im Unterricht", ein Thema, das nach wie vor auf ungebrochenes Interesse bei Lehrerinnen und Lehrern stösst.

Klänge – eines der schönsten Themen im Gymnasium, wird im Musik-, Biologie-, Mathematik- und Physikunterricht behandelt und in all diesen Fächern spielt die spektrale Zerlegung von Klangsignalen eine wichtige Rolle, naturgemäss aus verschiedenen Blickwinkeln betrachtet: beim Musiker sind es Grund- und Obertöne der Musikinstrumente, beim Biologen Stimm- und Gehörorgan, beim Mathematiker ist es die Fourierzerlegung und beim Physiker die Signalanalyse mit ihren weitreichenden Anwendungen in Wissenschaft und Technik, die Klänge zum fächerverbindenden Unterrichtsthema machen.

Der **Spektrumanalysator** sollte daher in keiner Schule fehlen, aber kommerzielle Geräte sind teuer und der Eigenbau ist nicht einfach. Doch, es ist unglaublich, aber wahr, Spektrumananlysatoren stehen in den Schulen zu Dutzenden herum, ohne als solche erkannt zu werden. Denn jeder moderne Computer ist mit einer Sound-Karte ausgerüstet, welche es möglich macht, ihn als schnellen und genauen Spektrumanalysator einzusetzen. Die Sound-Karte besitzt nämlich eine hervorragende Elektronik: so kann sie mit einer Abtastfrequenz von 44100 kHz in einer Auflösung von 16 bit auf 2 Kanälen gleichzeitig digitalisieren. Besitzer von herkömmlichen Laboradapter und Labor-Buskarten können davon nur träumen.

Wo ist aber die geeignete **Software** zu finden? Bekannt sind einige wenige Programme, die mehr für die Aufnahme und Verarbeitung von Musik, als für den Unterricht geeignet sind. Auf dem Mac, einem alten Vorreiter in Sachen Multimedia, sind die mir bekannten Programme etwas besser, auf Windows-Maschinen ist mir fast nichts Brauchbares bekannt. Was läge da nicht näher, als ein eigenes Programm zu

schreiben, das genau den persönlichen Ansprüchen und Gewohnheiten entspricht, und dies natürlich auf einer modernen graphikorientierten Benützeroberfläche, wäre da nicht der gefürchtete Aufwand an Programmierkenntnissen. Wir brauchen Menüs, Dialoge und graphische Darstellungen, sowie einen schnellen Zugriff auf die Sound-Karte in Echtzeit (man möchte ja logischerweise das Spektrum in Echtzeit sehen, d.h. mindestens 10 Spektren pro Sekunde darstellen können).

Die **Lösung** ist die Verwendung einer performanten Programmierumgebung, wie sie **C++/Champ** zur Verfügung stellt: speziell konzipiert für die Entwicklung von Programmen für den nicht-professionellen Programmierer, wie wir Lehrpersonen es ja sind. Der grosse Vorteil besteht darin, dass sich der Lernaufwand in Grenzen hält, da auf eine reiche Bibliothek an Objekten und Funktionen zurückgegriffen werden kann. So stellt Windows selbst einen Satz von Routinen zur Verfügung, mit denen man auf eine Sound-Karte **unabhängig** vom Fabrikat und ohne Kenntnisse der Karten-Kommandi und –Register zugreifen kann. Diese befinden sich in der Runtime-Library `MMSYSTEM.DLL`, die mit Windows automatisch installiert wird.

Um eine Idee zu vermitteln, seien einige Funktionen, die uns hier interessieren, erwähnt:

```
waveInOpen          Initialisiert die Sound-Karte
waveInStart         Startet die Aufnahme (kopiert in das RAM)
waveInStop          Stoppt die Aufnahme
waveInGetPosition   Liefert die aktuelle Aufnahmeposition
waveClose           Schliesst die Verbindung zur Sound-Karte
```

(Eine gute Beschreibung der Routinen mit Anwendungsbeispielen befindet sich in: Conger, Windows API, Waite Group 1993)

Die Verwendung dieser Funktionen setzt allerdings ein gewisses Verständnis für Windows-API-Programmierung voraus (API = Application Programming Interface, ist eine Programmsammlung für Betriebssystem-Aufrufe), denn es werden gewisse Low-Level-Parameter und -Strukturen benötigt.

Was läge näher, als einen kleinen sog. **Klassenwrapper** zu bauen, der die API-Funktionen "verpackt" und damit den Champ-Programmierer von den Low-Level-Funktionen "abschirmt". Mit dieser Klasse, von mir `CPAudio` genannt, wird die Verwendung der Sound-Karte quasi zum Kinderspiel: erstellt man eine Instanz der Klasse, so wird die Sound-Karte durch den Konstruktor initialisiert, also mit

```
CPAudio audio( samplingFreq, samplingTime );
```

erzeugt man ein Objekt, eben in bildhafter Vorstellung ein "Audio-Gerät", welches in der Lage ist, mit der `samplingFreq` während der `samplingTime` zu digitalisieren. Das Objekt ist im Sinn der objektorientierten Programmierung in der Lage, Mitteilungen zu empfangen, d.h. Methoden auszuführen. So kann man etwa mit

```
audio.startRecording( continuous )
```

die Aufnahme starten. Die Funktion besitzt einen einzigen fakultativen Parameter `continuous` vom Type `bool`. (Fakultative Parameter sind in den meisten Programmiersprachen unbekannt, da sie ja entbehrlich sind, sie ermöglichen aber einen eleganten Programmierstil!) Setzt man ihn auf `true`, so läuft die Aufnahme ununterbrochen weiter, d.h. die Daten werden in einen Memory Block geschrieben (der übrigens mehrere Megabyte umfassen kann) und nach dem Füllen beginnt die Aufnahme automatisch wieder von vorne.

Man kann die Karte auch "fragen", ob sie mit Aufnehmen beschäftigt ist. Dazu ruft man die Methode

```
audio.isRecording()
```

auf, welche `true` oder `false` zurückgibt.

Mit Hilfe solcher Klassenwrapper ist die Erstellung eines Fourier-analysators relativ einfach. Zwei verschiedene algorithmische Ideen drängen sich auf:

In einer Endlosschlaufe startet man jeweils die Aufnahme, um für eine Fast-Fourier-Transformation der Ordnung 2^{10} ein Paket von 1024 Datenwerten zu holen. Man wartet aber nicht auf das Ende der Aufnahme (was bei einer Samplingfrequenz von 22050 Hz ca. 50 ms dauert), sondern führt in der Aufnahmezeit die Transformation des letzten Datenpakets und seine graphische Darstellung aus. Je nach der Geschwindigkeit des Rechners benötigt man dazu nämlich weniger als 50 ms, so dass man sogar noch auf das Ende der Aufnahme warten muss. Damit erhält man im besten Fall alle 50 ms eine Spektralanalyse - nicht schlecht!

Es geht aber noch schneller: Man lässt die Sound-Karte ununterbrochen Datenwerte ins Memory lesen und holt in der Endlosschlaufe jeweils die aktuelle Position. Von dort aus liest man die letzten 1024 Werte aus dem Memory und analysiert diese. Da die Analyse/Darstellung auf einem

schnellen Rechner in etwa 20 ms erledigt werden kann, erhält man bei diesem Vorgehen sich zeitlich überschneidende Datenpakete.

Dieses schnelle Verfahren entspricht eigentlich dem modernen Konzept des **Multithreading**: ein Thread führt die Datenaufnahme durch, der andere die Berechnung der Daten.

Das hier eingesetzte Klassenkonzept eignet sich nicht nur hervorragend für "Geräte", wie Sound-Karten, Com- und Lpt-Ports, sondern auch für Algorithmen. So habe ich eine kleine Klasse `Fft` geschrieben, welche den FFT-Algorithmus verpackt. Konstruiert man mit

```
Fft fft( 1024 );
```

ein Objekt, so besitzt dieses Ding gescheite Methoden. So führt der Aufruf

```
fft.transform( ary );
```

von den Daten im übergebenen Array eine 1024-Punkte-FFT durch und liefert das Resultat in den Arrayelementen 0..512 zurück.

Auch für die graphische Darstellung ist eine Klasse geeignet, schon deswegen, weil man ja gewöhnlich mehrere Graphikfenster, etwa eines für die Zeitfunktion und eines für das Spektrum benötigt. Die eingebaute Champ-Klasse `CPWindow` besitzt zwar bereits viele gute Methoden, aber ihr fehlt beispielsweise die Möglichkeit, ein Koordinatensystem darzustellen. Hier kommt eines der besten Konzepte der objektorientierten Programmierung zum Zug: man **erweitert** eine bestehende Klasse durch Klassenableitung: das neue Objekt besitzt alle Methoden der Basisklasse (falls man diese nicht **überschreibt**) und zusätzlich die neu definierten. Die von `CPWindow` abgeleitete Klasse `CroPix` (für Cathode-Ray-Oscilloscope with Pixel Coordinates) erhält so zum Beispiel die Methode `drawGrid,` welche das Koordinatensystem einzeichnet.

Stellt man dem Programmierer all diese Klassen als Bibliothek zur Verfügung, so kann er sich, wie vor etlichen Jahren bei der Programmierung mit Basic, Pascal oder Fortran auf IBM-, Atari-, Amiga-, Commodore oder DEC-Rechnern ganz auf den Algorithmus seines Problems konzentrieren. Für den Spektrumanalysator, der nach dem vorhin erstgenannten Algorithmus arbeitet, sieht dies so aus:

```
   CroPix timeCro ( ... );
   CroPix fftCro ( ... );

   CPFft fft( fftOrder );
   CPAudio audio( samplingFreq, samplingTime );

   bool first = true;
   do
   {
      switch ( state )
      {
         case stopped:
            audio.stopRecording();
            first = true;
            CP::yield();
            break;

         case recording:
            if ( first )
            {
               audio.startRecording();
               first = false;
               break;
            }

            while ( audio.isRecording() ) {}  // Do nothing

// Get data from memory
            for ( i = 0; i < fftOrder; i++ )
               ary[i] = audio.aryData[i];

            // and restart recording immediately
            audio.startRecording();

            timeCro.showTrace( ary );
            fft.transform( ary );
            fftCro.showTrace( ary );
            break;
      }
   } while ( state != aborting );
}
```

Die Verdeckung durch Klassenbildung ist auch in der Grundausbildung durchaus legitim und nicht etwa eine didaktisch fragwürdige Verheimlichung der Funktionsweise. Wer es genauer wissen will und die Zeit und Lust dazu aufbringt, kann ja immer noch die tieferen Schichten hinterfragen, bis hin zur Assemblerprogrammierung und Halbleiter-elektronik.

(Der komplette Source-Code ist auf dem WEB erhältlich, und zwar unter **http://www.kl.unibe.ch/kl/lab/champ** .)

25 ms-Zeitsignal und sein10 kHz-Spektrum in Echtzeit des russischen Sängers Vladimir Vissotski.