Demo-Programmierung unter Windows 95/NT

Grafik mit System

Die Programmierung schneller Grafikeffekte unter Windows ist keine Zauberei. Schaffen Sie mit einer Bibliothek die Grundlagen dazu.

CARSTEN DACHSBACHER/
NILS PIPENBRINCK

Multimedia ist seit vielen Jahren ein Schlagwort der Computerindustrie. Aber nicht nur Konzerne, sondern auch Künstler haben das Zusammenspiel von Grafik- und Soundsystem für sich entdeckt. Das Ergebnis ihrer Arbeit sind selbstablaufende Multimediapräsentationen, sogenannte Demos.

Eine Demo entsteht in einem kreativen Prozeß, bei dem Programmierer, Grafiker und Musiker ihre Erfahrungen und ihr Talent einbringen. In dieser neuen Rubrik zeigen wir Ihnen das nötige Handwerkszeug, um mitreißende Multimedia-Clips zu entwickeln.

DIE GRAFIK für unser Beispielprogramm enthält 256 Farben.

Dabei will PC Magazin Ihrer kreativen Ader kräftig auf die Sprünge helfen: Wir bieten Ihnen als Grundlage lauffähige Listings an, die Sie persönlich weiterentwickeln können. Aus Platzgründen drucken wir nur die wichtigsten Routinen ab. Die kompletten Programme können Sie sich von der Heft-CD, von der Databox oder aus dem Internet-Angebot des PC Magazin unter

`www.pc-magazin.de`

auf Ihren Rechner laden.

Step by Step

Wie schreiben Sie nun eine professionelle Demo? Auf den ersten Blick könnten Sie versucht sein, eine komplette Animation zu berechnen, zu vertonen und das Ergebnis als Video zu speichern. Das mag zwar funktionieren, aber die Faszination von Demos liegt eigentlich darin, den Computer so weit wie möglich auszureizen, keine Ressourcen zu verschwenden und möglichst alles in Echtzeit zu berechnen.

Das ist nicht immer ganz einfach, aber erlernbar. Mit PC Underground führen wir Sie jeden Monat ein Stück weiter in die Geheimnisse der Profis ein. Bereits nach kurzer Zeit haben Sie eine komplette Demo geschrieben. In dieser Ausgabe des PC Magazin legen Sie mit einem Grafiksystem für die Windows-95/NT-Plattform den Grundstein dazu. Außerdem lernen Sie anhand eines einfachen Effektes die ersten Tricks der Demo-Programmierer kennen.

Sie benötigen lediglich etwas C-Kenntnisse und für einige zukünftige Routinen eventuell – aber nicht unbedingt notwendig – Interesse an Assembler-Programmierung. Als Compiler eignen sich gleichermaßen die Produkte von Borland, Microsoft, Watcom oder Intel.

Sie können auf Assembler-Code sogar vollständig verzichten, da die heutigen C-Compiler durchweg gute Ergebnisse produzieren. Besonders erwähnenswert ist hier der Intel-C/C++-Compiler, dessen Code qualitativ oft an handoptimierten heranreicht. In den Beispielprogrammen bietet Ihnen PC Magazin zu jeder Assembler-Routine auch das entsprechende C-Pendant an.

32-Bit Protected Mode

Als Entwicklungsplattform dient Win32, die Demo läuft also unter Windows 95 und NT. Im Hinblick auf die später in dieser Reihe verwendeten Assembler-Module ist es sinnvoll, die Eigenheiten des Protected Mode und des Windows-Speichermodells zu erläutern.

Als noch MS-DOS und Windows 3.1x die PC-Welt regierten, liefen Programme im 16-Bit-Real-Mode und Protected Mode. 16 Bit deshalb, weil alle Prozessorregister 16 Bit breit waren.

Der Nachteil daran war, daß ein 16 Bit breites Register nur 64 KByte (= 65 536) Speicherblöcke adressieren konnte. Die Blöcke wurden daher mit einem Registerpaar angesprochen, nämlich mit einer Segment- und einer Offset-Adresse. Natürlich konnte ein Programm mehrere dieser Blöcke anfordern, aber bei großen Datenstrukturen war dies sehr unpraktisch.

Mit der Einführung des 386-Prozessors von Intel wurden alle Register auf 32 Bit erweitert. Anfangs war es nicht einfach, diese wirklich effektiv zu

DATEN – STÄNDIG AKTUELL

Alle Daten (Listings und Bilder) zu unserem Beispielprogramm liegen auf der Homepage des PC Magazin für Sie bereit:

`www.pc-magazin.de`

Sie finden diese Daten deshalb in unserem Online-Programm, da wir sie ständig aktualisieren und mit jeder neuen Ausgabe weiter ausbauen.

PILGERFAHRT NACH FALLINGBOSTEL

Schwer bepackte Jugendliche pilgerten Karfreitag dieses Jahres nach Fallingbostel in die Lüneburger Heide. Doch nicht, um Buße zu tun oder die Ostermesse zu besuchen. Ihr Ziel war die Mekka & Symposium 98, Deutschlands bisher wohl größte Demo-Party.

PARTYATMOSPHÄRE wie im Raumfahrtzentrum

Mit Demos sind hier nicht Testversionen kommerzieller Software gemeint. Diese Demos bestehen vielmehr aus einer Mischung verschiedener Effekte und Grafiken, hinterlegt mit Musik. In ihrer Art erinnern sie oft an moderne Musikvideos, wie sie bei MTV oder VIVA laufen.

Das Schreiben von Demos entwickelte sich in den 90er Jahren zu einer richtigen Jugendkultur mit eigener Szene und eben auch großen Partys. Heutige Programme enthalten mehrere MByte an Code und Daten, weshalb sich verschiedene Spezialisten die Arbeit aufteilen.

Grafiker entwerfen kleine Logos und Schriftsätze sowie hochauflösende Bilder und Animationen. Handarbeit besitzt großen Wert, der Einsatz von Scannern ist verpönt. Elektronische Hilfsmittel wie Filter oder Raytracer schaffen dafür oft atemberaubende Effekte.

Zum visuellen Eindruck fügen Musiker noch das Klangerlebnis hinzu. Sie reihen Samples zu einzelnen Tracks aneinander und komponieren daraus ihre Songs.

Die Programmierer – in der Szene Coder genannt – schreiben den eigentlichen Quellcode der Demos. Sie implementieren Laderoutinen für die Grafiken, entwickeln immer neue überraschende Grafikeffekte und kämpfen dabei um jedes Quentchen Geschwindigkeit, das dem Prozessor zu entlocken ist. Schließlich sorgen sie auch dafür, daß die Songs synchron mit dem selbstablaufenden Grafikspektakel einsetzen.

Die Mekka & Symposium '98 war eine einmalige Gelegenheit, Grafikprogrammierer und Musiker aus dem In- und Ausland zu treffen. Die 72 Stunden dauernde Party bot den rund 850 fast durchweg männlichen Besuchern genug Zeit, um neue Ideen zu besprechen und Wissen auszutauschen. Der Blick über den Tellerrand war erlaubt, neben der PC-Welt gab sich auch die Commodore-Gemeinde der C64er und Amigas die Ehre.

Im Gepäck durfte ein gutbestückter Rechner natürlich nicht fehlen, und so bot die angemietete Mehrzweckhalle bald das Bild eines Raumfahrt-Kontrollzentrums.

Per Ethernet entstand ein großes Partynetzwerk, über das Daten und Programme, aber auch schon mal das eine oder andere Netzwerkspiel flossen. Den allgemeinen Lärmpegel steigerten einige Teilnehmer zusätzlich mit einer Stereoanlage oder einem Synthesizer.

Gelegentlich führte der hohe Energieverbrauch der mitgebrachten Kaffeemaschinen, Mikrowellen und Waffeleisen zu Stromausfällen, die zu kurzen Pausen an der Pommes-Bude vor der Halle genutzt wurden.

Hauptereignisse der Party waren Wettbewerbe, zu denen jeder Besucher seine Demos, Spiele, Grafiken und Musikstücke einreichen konnte. Die Ergebnisse wurden dann auf eine 8 x 6 Meter große Leinwand projeziert und von einer baßkräftigen Sound-Anlage unterstützt. Jeder im Saal konnte so die Präsentation verfolgen, und wer sich im benachbarten Schlafzelt zur Ruhe gelegt hatten, spürte noch die Bässe in den Knochen.

Ab und zu ging ein bewunderndes Raunen durch die Halle, manchmal tobte der Applaus. So auch bei der Siegergrafik von Cyclone/Abyss mit dem geheimnisvollen Titel *Hänsel und Gretel im Paradies des blutigen Todes*.

Bei den PC-Demos trug die Gruppe Matrix mit dem aufwendig gestalteten *Fulcrum* einen überragenden Sieg davon. Die Wahl der Gewinner erfolgte in bester Basisdemokratie durch alle Anwesenden.

Auf der Heft-CD finden Sie neben den beiden Gewinnern weitere Demos, Spiele, Grafiken und Sounddateien im *mod*-Format der Mekka & Symposium '98. Auf mehr Material verweist die von den Veranstaltern betriebene WWW-Seite

```
http://ms.demo.org
```

Eine sehr gute Übersicht über international anstehende Partytermine bietet

```
www.hornet.org/ha/pages/
➥calendar.html
```

Eine kleine Kuriosität am Rande war die erste Demo für den Nintendo Gameboy, welche auf einem Emulator lief. Auf der Video-Leinwand sorgte die sehr pixelige Darstellung aber eher für eine anerkennende Würdigung als für gebanntes Staunen.

Der geplante Demo-Wettbewerb für Windows 95 fiel mangels Beteiligung aus. Das mag vor allem daran liegen, daß viele Coder die Hardware gerne direkt unter MS-DOS ansprechen, was Windows nun mal nicht zuläßt. Windows bietet dafür andere Annehmlichkeiten für die Programmierer, und moderne Prozessoren halten längst die geforderte Leistung bereit. So ist es wohl nur eine Frage der Zeit, wann auch anspruchsvolle Demos für Windows 95/NT das Licht der Welt erblicken.

HÄNSEL UND GRETEL im Paradies des blutigen Todes

Vielleicht sind Sie einer der ersten, die eine Demo für Windows 95/NT produzieren. Denn PC Magazin startet in dieser Ausgabe die neue Rubrik PC Underground, die an die Stelle des 1000-Zeilen-Wettbewerbs tritt. In den ersten Folgen erfahren Sie von zwei Codern der Gruppe Cubic&$een Tricks zur Grafikprogrammierung unter 32-Bit-Windows. Dabei entwickeln Sie schrittweise eine komplette Demo mit allem, was dazugehört.

RÜDIGER PEIN/BM

nutzen: Weder MS-DOS noch Windows 3.1x waren darauf ausgelegt, Programme im 32-Bit-Protected-Mode auszuführen. Seit Win32s, Windows 95 und NT stellt dies kein Problem mehr dar. Auch unter DOS umgehen Sie dieses Problem mit dem Einsatz sogenannte DOS-Extender.

Mit 32 Bit breiten Registern adressieren Sie nun einen Adreßbereich von 4 GByte linear. Das bedeutet, daß Sie mehr als 64 KByte Speicher an einem Stück anfordern und ansprechen können. Programme werden dadurch einfacher und übersichtlicher.

Windows-Programmierung

Beim Schreiben einer Demo möchten Sie sich nicht jedesmal um die Initialisierung eines Fensters kümmern. Diese Aufgaben und häufig gebrauchte Funktionen fassen Sie deshalb in einer Bibliothek zusammen.

Ihre Aufgabe ist es nun, solch ein Basissystem auf der Grundlage der Windows-GDI (Graphics Device Interface) zu schreiben. Diese Schnittstelle kann Grafiken laden, Farbmanipulationen an Bildern durchführen und berechnete Bilder entweder in einem Fenster oder im Vollbildmodus darstellen. Zudem kann sie den Ablauf einer Demo unabhängig von der Geschwindigkeit des Rechners steuern.

Zunächst einmal sind Demos keine interaktiven Programme. Windows selbst ist aber darauf ausgelegt, nicht nur Inhalte in Fenstern darzustellen, sondern auch auf Eingaben des Benutzers zu reagieren. Da Sie in unserem Fall nicht auf spezielle Benutzereingaben achten müssen, genügt ein einfacher Windows-Startup, der für die eigentliche Demo alles unsichtbar erledigt.

Dieser Startup ist so allgemein und unabhängig, daß man ihn durch einen anderen ersetzen könnte. Somit wäre ein und dieselbe Demo auf anderen Betriebssystemen lauffähig.

Eine normale Anwendung erzeugt prinzipiell ein Fenster mit einem Event Handler, also einer Prozedur, die auf Eingaben des Anwenders reagiert. Danach arbeitet die Anwendung bis zum Schließen des Fensters eine Hauptschleife ab.

Diese Schleife wartet auf Benutzereingaben oder Nachrichten und leitet diese dann an das zuständige Fenster beziehungsweise dessen Event Handler weiter. Diese führen dann vom Programmierer bestimmte Routinen aus. Da die Interaktion des Benutzers bei einer Demo auf das vorzeitige Beenden des Programms beschränkt sein soll, ist der Event Handler sehr einfach.

Da Windows ein Multitasking-Betriebssystem ist, können mehrere Programme gleichzeitig ablaufen. Die Verwaltung der einzelnen Programme und Programmabläufe (Threads) übernimmt dabei der Windows-Kernel (Betriebssystemkern). Sie brauchen sich also als Programmierer darüber keine Gedanken zu machen. Bei der Verwendung mehrerer Threads spricht man von Multithreading.

Für eine Demo ist Multithreading die ideale Lösung. Sie teilen das Programm einfach in zwei Threads auf: Der erste Thread ist für die Windows-Messages zuständig, während der zweite die eigentliche Demo und deren Ablauf darstellt.

Aufbau des Basissystems

Das Basissystem der Demo sollte so unabhängig vom Betriebssystem sein, daß Sie sich als Programmierer nicht mehr im geringsten mit der Windows-Programmierung beschäftigen müssen. Sie werden bei der Entwicklung Ihrer Demos nur noch zwei Funktionen implementieren:

- Die Funktion

```
BOOL demoinit(void)
```

enthält Initialisierungs-Routinen, zum Beispiel, um Grafiken zu laden oder Musik zu starten. Nach erfolgreicher Ausführung geben Sie in *demoinit* den Wert *1* (für *true*) zurück.

- In die Funktion

```
void demomain(void)
```

schreiben Sie den Ablauf der Demo, also den eigentlichen Programmcode. Dabei stehen Ihnen verschiedene Bibliotheksfunktionen zur Verfügung:

```
unsigned long GetDemoTime(void)
```

GetDemoTime gibt die seit dem Demostart verstrichene Zeit in Millisekunden wieder. Dies ist notwendig, wenn ein Effekt unabhängig von der Rechnergeschwindigkeit ablaufen soll. Im Beispiel dieser Ausgabe ist es die Lichtquelle, deren Position abhängig von der verstrichenen Zeit bestimmt wird.

```
unsigned short ColorCode(
  int r,int g,int b)
```

ColorCode liefert für das Wertetripel (*r*, *g* und *b*) den Farbwert des in der Demo verwendeten Farbmodells.

```
void BlitGraphic(void *buf)
```

Mit *BlitGraphic* übergeben sie einen Zeiger auf ein darzustellendes Bild.

```
int bmp_load(char *name,
  bitmaptype &bitmap)
```

Mit *bmp_load* laden Sie *bmp*-Dateien für den Gebrauch in Ihre Demo. Der Typ *bitmaptype* ist selbstdefiniert und enthält Breite, Höhe und Zeiger auf die Bilddaten. Den genauen Aufbau entnehmen Sie dem Quelltext.

```
void bmp_make16bitpalette(
  bitmaptype &bitmap)
```

bmp_make16bitpalette konvertiert im Falle einer 256-Farben-Bitmap die Farbpalette in das in der Demo verwendete Farbmodell (siehe Textbox oben).

```
void bmp_free(
  bitmaptype &bitmap)
```

bmp_free gibt den Speicher einer geladenen Bitmap-Datei wieder frei.

Außerdem stellt die Bibliothek noch einige Tabellen und Konstanten bereit, deren Bedeutung Sie anhand des Beispiels erfahren.

DAS VERWENDETE FARBMODELL

Das in diesem Projekt verwendete Farbmodell entspricht jenem, das die meisten Grafikkarten in HiColor-Modi verwenden. Dazu steht für jedes Pixel ein 16 Bit breiter Wert zur Verfügung, in dem jeweils ein Bereich der Bits den Rot-, Grün- und Blauanteil (RGB) einer Farbe darstellt.

Die Bereichsgrößen sind jeweils 5 Bit für Rot und Blau und 6 Bit für Grün. Der Aufbau sieht also folgendermaßen aus:

```
RRRRRGGGGGGBBBBB
```

Drei Tabellen – für jede Primärfarbe eine – mit jeweils 256 Einträgen vereinfachen die Handhabung. Das sind natürlich mehr als notwendig: Mit 5 Bit lassen sich maximal $2^5 = 32$ verschiedene Abstufungen erzeugen. Viele der 256 Einträge enthalten daher den gleichen Wert. Dennoch sind diese Tabellen sehr nützlich, sie vereinfachen und beschleunigen grafische Effekte mit Farbmanipulationen.

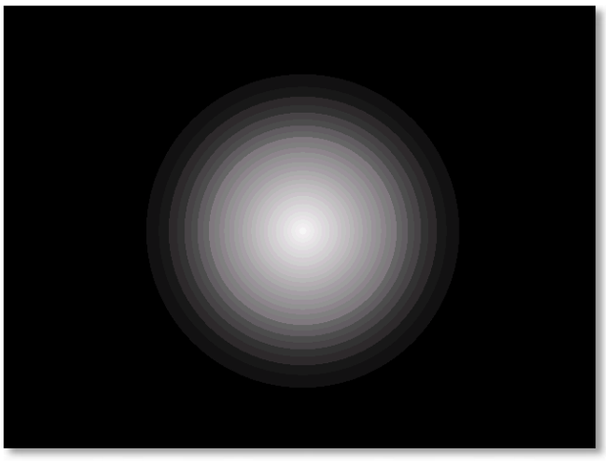
Den Farbwert eines satten Orangetons erhalten Sie etwa durch Auslesen der Einträge *230, 100* und *20* aus den Tabellen und einer bitweisen Oder-Verknüpfung:

```
unsigned short farbwert =
➥Rtab[230] | Gtab[100] |
➥Btab[20];
```

Die Farben ergeben sich durch additive Farbmischung, das heißt größere Werte (bis einschließlich *255*) ergeben einen helleren Anteil der Primärfarbe. Das Wertetripel *(255,255,255)* ergibt also Reinweiß, *(0,0,0)* die Farbe Schwarz.

Implementierung des Basissystems

Jedes Windows-Programm startet mit der Routine *WinMain*. Diese soll nun eine Bitmap-Info erzeugen, um eine Grafik mit einem bestimmten – vom Programmierer festgelegten – Aufbau in einem Fenster darzustellen. Außerdem soll sie ein Fenster öffnen, in dem die Demo angezeigt wird.



DIESE LIGHTMAP erzeugt den Lichteffekt.

Die Prozedur *InitGraphic(void)* erzeugt diese Bitmap-Info. Sie fordert dazu eine Variable des Typs *BITMAPINFO* an und trägt Breite und Höhe des Fensters sowie das verwendete Farbmodell ein.

Das Erzeugen des Fensters ist Aufgabe der Prozedur *InitDemoWindow*. Dazu registriert sie eine neue Fensterklasse, legt mit *CreateWindowEx* eine Instanz davon an und stellt sie mit *ShowWindow* dar. Die Definition einer Fensterklasse enthält zum Beispiel das Erscheinungsbild eines Fensters mit den vorhandenen Buttons, dem Icon und dem Mauscursor. Zudem speichert sie einen Verweis auf die Funktion, die die Nachrichten dieses Fensters verarbeitet.

Wenn das Fenster sichtbar ist, wird noch der Device Context in einer Variable gesichert. Um Grafiken im Fenster darzustellen, beziehen Sie sich von nun an auf diesen Verweis.

Außerdem setzt *InitDemoWindow* die Priorität des ersten Threads, der die Nachrichten an das Fenster verarbeitet, auf den niedrigsten Wert. Dadurch können Sie dem zweiten Thread, der nach *InitDemoWindow* startet, eine hohe Priorität geben. Sie gewähren somit dem Programmcode für die Demo mehr Rechenzeit. Nach den Initialisierungsarbeiten geht *WinMain* in die Message-Schleife über, die die Nachrichten an die Message-Funktion weiterleitet und das Programmende abwartet.

Die Message-Funktion des Demofensters ist wie bereits erwähnt sehr einfach. Diese Funktion wird immer aufgerufen, wenn eine Nachricht an das Fenster gesandt wird. Die Nachrichten stammen entweder vom Windows-System selbst oder vom Benutzer.

Für das Demofenster sind nun zwei Nachrichten interessant: *WM_DESTROY* zeigt an, daß das Fenster geschlossen werden soll, *WM_KEYDOWN* signalisiert einen Tastendruck. In beiden Fällen soll die Demo beendet werden. Dazu geben Sie den Device Context wieder frei und teilen der Message-Schleife über *PostQuitMessage* mit, daß das Programm beendet werden soll.

Alle anderen Nachrichten, die nicht speziell behandelt werden müssen, übergeben Sie an *DefWindowProc*. Diese Prozedur verwaltet Nachrichten wie das Verschieben des Fensters oder das Öffnen des System-Menüs und enthält Standardbehandlungsroutinen für die meisten Nachrichten.

Der *BlitGraphic*-Funktion übergeben Sie einen Zeiger auf eine Bitmap, deren Farbwerte dem verwendeten Farbmodell entsprechen. Sie ruft die benötigten Windows-GDI-Funktionen auf, um eine Device Independent Bitmap in einem Fenster darzustellen. Dabei handelt es sich um eine Bitmap, die unabhängig vom Bildschirmmodus ist, in dem sich die Grafikkarte befindet. Muß das Bild nicht skaliert werden, kommt dafür *SetDIBitsToDevice* in Frage, für alle anderen Fälle erledigt dies die Funktion *StretchDIBits*.

Der erste Demoeffekt

Bevor Sie den ersten Demoeffekt programmieren, zeigt Ihnen folgendes Beispiel den Umgang mit der Grafikbibliothek:

```
#include „demo.h“

bitmaptype bmp;

BOOL demoinit(void)
{
  bmp_load(„BACK256.BMP“, bmp);
  bmp_make16bitpalette(bmp);
  return 1;
}

void demomain(void)
{
  unsigned short screen[
    SCREEN_X*SCREEN_Y];

  for (int i=0;
       i<SCREEN_X*SCREEN_Y;
       i++)
    screen[i]=bmp.sColors[
      bmp.cBitmap[i]];

  BlitGraphic(screen);
  while (DemoRunning);
}
```

Die Funktion *demoinit* lädt eine 256-Farben-Bitmap in Fenstergröße und bereitet eine Palette fürs Farbmodell vor.

Mit Hilfe dieser in *bmp.sColors* gespeicherten Palette setzt nun *demomain* jedes Pixel in den virtuellen Bildschirm *screen*. Die Funktion *BlitGraphic* stellt das fertige Bild im Fenster dar, die nachfolgende While-Schleife wartet, bis das Demosystem das Ende signalisiert. In diesem Fall enthält die Variable *DemoRunning* den Wert *0* (für *false*).

DIE ÜBERLAGERUNG von Bitmap und Lightmap sieht so aus.

Als Beispiel für die Nutzung des Basissystems der Demo laden Sie eine *bmp*-Datei und stellen sie mit einer darüber schwebenden Lichtquelle dar. Die folgenden Schritte implemetieren Sie in der Funktion *demoinit*:

Zunächst laden Sie mit *bmp_load* eine Bitmap mit 320 x 240 Bildpunkten und 256 Farben in den Speicher. Nun soll eine Lichtquelle über das Bild wandern. Die Bitmap soll nahe der Lichtquelle hell sein und mit zunehmender Entfernung dunkler werden.

Sie arbeiten im folgenden mit 32 Helligkeitsstufen, und für alle 256 Farben des Bildes berechnen Sie in einer sogenannten Shading-Tabelle die 32 Abstufungen vor. Dazu multiplizieren Sie die Rot-, Grün- und Blauwerte mit der Helligkeitsstufe und teilen das Ergebnis durch 12. Ist die Helligkeitsstufe größer als 24, so addieren Sie noch einen Wert hinzu. Durch eigene Versuche bestimmen Sie ähnlich geeignete Werte – diese bewirken dann ein leicht geändertes Erscheinungbild.

Nun berechnen Sie noch eine sogenannte Lightmap. Das ist eine Bitmap, die viermal so groß ist wie das Original, also 640 x 480 Bildpunkte. Jeder Bildpunkt der Lightmap enthält den Helligkeitswert, der von seiner Entfernung zum Mittelpunkt (320,240) der Lightmap abhängt. Diesen Wert bestimmen Sie durch den Sinus des Abstands. Auch diese Formel entstammt empirischen Versuchen und erlaubt leichte Modifikationen.

Die Berechnung der Einzelbilder erfolgt nun in der *demomain*-Funktion: Ein neues Bild berechnen Sie, indem Sie die Lightmap über der Bitmap verschieben und dann beide überlagern.

Bei diesem Verfahren führen zu große Verschiebungen der Lightmap zu Stellen, an denen sie nicht mehr mit der Bitmap überlappt. Diese Bereiche sind aber ohnehin sehr weit von der Lichtquelle entfernt, die sich ja in der Mitte der Lightmap befindet. Deswegen dürfen Sie dort getrost den gleichen Helligkeitswert wie am Rand der Lightmap annehmen.

Da die X-Koordinaten der Lightmap für alle Spalten des Endbildes gleich sind (dasselbe gilt für die Y-Koordinaten bezüglich der Zeilen), berechnen Sie sie vor dem eigentlichen Zeichnen vor und speichern sie in *pos_x[]* (bzw. *pos_y[]*). Für jedes Pixel des Endbildes lesen Sie den entsprechenden Wert der Lightmap und den Pixel des Originalbildes aus und bestimmen mit Hilfe der Shading-Tabelle den neuen Farbwert. Diese neuen Farbwerte stehen dann in *screen* und gelangen mit *BlitGraphic* zur Darstellung.

Sie sehen, wie die verschiedenen vorberechneten Tabellen (Lightmap, Shading-Tabelle und *pos_x[]* bzw. *pos_y[]*) dazu beitragen, die Animation flüssig laufen zu lassen. Natürlich könnten Sie auch in der Hauptschleife für jeden Pixel den Abstand zur Lichtquelle und damit den Helligkeitswert bestimmen, dann die Farbe des Pixels der Bitmap auslesen und anhand dieser RGB-Werte eine neue Farbe bestimmen. Nur würde das Ergebnis nicht mehr einer Bewegung, sondern mehr einer Slideshow ähnlich sehen.

Ausblick

Den ersten Teil mit der Vorstellung des Grafiksystems und den ersten Einblicken in die Demoprogrammierung haben Sie nun gemeistert. In den nächsten beiden Ausgaben des PC Magazin entwickeln Sie eine 3D-Engine, mit der Sie komplexe animierte Szenen in Echtzeit berechnen.

Fertige Demos der beiden Autoren können Sie im Internet unter

www.cubic.org

herunterladen. Dort finden Sie im Abschnitt *Gallery* Demos für DOS und Windows. PEI/BM

1 Ausschnitt aus Demosys.cpp

```
static BOOL InitGraphic( void )
{
    // Vorbereiten eines BitmapInfoHeaders für
    // alle Grafikausgaben.
    int bisize = sizeof( BITMAPINFOHEADER );
    bitmapinfo = (BITMAPINFO *)malloc( bisize + 12 );
    ZeroMemory( &bitmapinfo->bmiHeader, bisize );

    // BitmapInfoHeader eines normalen 16-Bit-Bitmaps,
    // wie wir es brauchen, erzeugen
    bitmapinfo->bmiHeader.biSize         = bisize;
    bitmapinfo->bmiHeader.biWidth        = SCREEN_X;
    bitmapinfo->bmiHeader.biHeight       = -SCREEN_Y;
    bitmapinfo->bmiHeader.biPlanes       = 1;
    bitmapinfo->bmiHeader.biBitCount     = 16;
    bitmapinfo->bmiHeader.biCompression = BI_BITFIELDS;
    // Farb-Felder des 16-Bit Bitmaps setzen.
    ((long*) &bitmapinfo->bmiColors)[0] = 0xF800;
    ((long*) &bitmapinfo->bmiColors)[1] = 0x07E0;
    ((long*) &bitmapinfo->bmiColors)[2] = 0x001F;

    // Berechnen der 16 Bit Farbtabelle
    for ( int i = 0; i < 256; i++ )
    {
        Rtab[ i ] = ColorCode( i, 0, 0 );
        Gtab[ i ] = ColorCode( 0, i, 0 );
        Btab[ i ] = ColorCode( 0, 0, i );
    }
    return 1;
}

void BlitGraphic( void *buf )
{
    if ( !DemoHDC ) return;

    switch ( Fenster_Modus )
    {
    case FENSTER:
      SetDIBitsToDevice( DemoHDC, 0, 0, SCREEN_X, SCREEN_Y,
                         0, 0, 0, SCREEN_Y, buf, bitmapinfo,
                         DIB_RGB_COLORS );
      break;
    default:
      RECT r;
      GetClientRect( DemoHWND, &r );
      StretchDIBits( DemoHDC, 0, 0, r.right, r.bottom, 0, 0,
                     SCREEN_X, SCREEN_Y, buf, bitmapinfo,
                     DIB_RGB_COLORS, SRCCOPY );
      break;
    }
}

long CALLBACK WindowProc( HWND hWnd, UINT message,
                          WPARAM wParam, LPARAM lParam )
{
    // Dies ist die Message Funktion des Demo-Fensters.
    // Da das Fenster selbst sogut wie keine Funktionalität
    // haben muß, ist diese Funktion sehr kurz
    switch (message)
    {
        case WM_DESTROY:
        case WM_KEYDOWN:
            ReleaseDC( DemoHWND, DemoHDC );
            DemoHDC = 0;
            DemoRunning = 0;
            PostQuitMessage( 0 );
            break;
    }
    return DefWindowProc( hWnd, message, wParam, lParam );
}

int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                   LPSTR lpCmdLine, int nCmdShow)
{
    unsigned long ThreadID;
    MSG         message;

    // Tabellen und Strukturen der Bibliothek initialisieren
    if ( !InitGraphic() ) return 0;

    // Hier darf das Demo sich erst einmal initialisieren
    if ( !demoinit() ) return 0;

    // Fenster Erzeugen und Zeigen
    if ( !InitDemoWindow( nCmdShow, hInstance ) ) return 0;

    // Jetzt kann nichts mehr schiefgehen
    // der Haupt-Thread des Demos kann gestartet werden
```