AUF DER HEFT-CD

Demo-Programmierung unter Windows 95/NT

Auf direktem Wege

Programmieren Sie **schnelle Grafik mit DirectX**, und nutzen Sie Ihre Demos als Bildschirmschoner.

CARSTEN DACHSBACHER/
NILS PIPENBRINCK

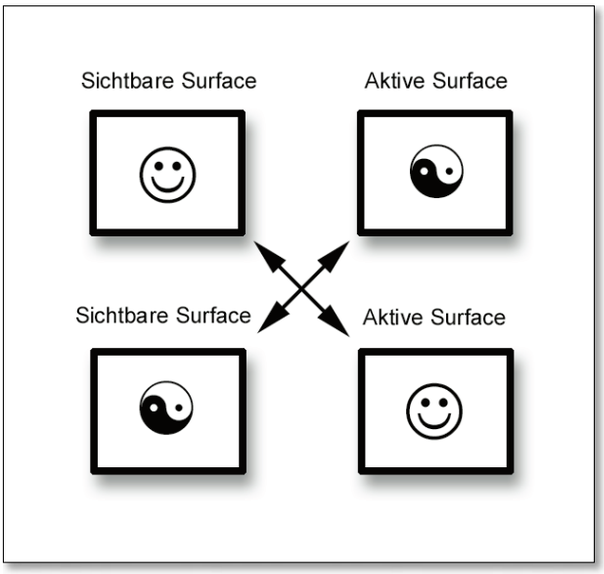
Windows-Anwendungen stellen Grafiken meist über das Graphics Device Interface (GDI) dar. Diese Schnittstelle enthält ein sehr aufwendiges System zur Fensterverwaltung. Außerdem bietet sie viele Funktionen, um einfache grafische Objekte wie Linien und Rechtecke zu zeichnen.

Bei der Demo-Programmierung liegt Ihr Interesse aber weniger in der Fensterverwaltung als vielmehr in maximaler Geschwindigkeit. Um dem allgemeinen Wunsch nach mehr Grafik-Power gerecht zu werden, hat Microsoft mit dem Erscheinen von Windows 95 eine zweite Schnittstelle für schnellere Grafik geschaffen: Sie heißt DirectX und erfreut sich vor allem bei Spielen großer Beliebtheit.

Die Struktur von DirectX wollen wir uns genauer ansehen. Als Anwendungsbeispiel schreiben Sie eine Grafikbibliothek, die einen echten Vollbildmodus unter DirectDraw – einem Bestandteil von DirectX – bietet. Sie verwenden dabei relativ einfache Aufrufe, die mit DirectX ab Version 3.0 zusammenarbeiten (neuere Versionen sind abwärtskompatibel).

Das Besondere an DirectX

Mit DirectX begann Microsoft, ein neues Modell für Programmierschnittstellen zu verwenden. Es ist das Component Object Model (COM). Die alte, auf Funktionen basierende Schnittstelle sollte durch eine objektorientierte ersetzt werden. Dabei traten zwei technische Probleme auf: Zum einen booten nicht alle Windows-Programmiersprachen objektorientierte Funktionen, zum anderen lassen sich mit DLL-Bibliotheken keine Objekte exportieren.



BEIM PAGE-FLIPPING vertauscht DirectDraw einfach die sichtbare mit der unsichtbaren Oberfläche.

Für den C++-Programmierer sieht der DirectX-Quelltext objektorientiert aus. In Wirklichkeit täuschen Makros nur die Objektorientierung vor. Bereiten Sie sich daher auf außergewöhnliche Fehlermeldungen Ihres C-Compilers vor, denn dieser sieht den Code anders, als Sie ihn eingeben.

DirectX ist die der Hardware am nächsten stehende Schnittstelle, die die Windows-API zu bieten hat. Daher ist sie besonders fehleranfällig. Sie sollten stets den Rückgabewert von DirectX-Funktionen überprüfen: Wenn Sie einen Fehler ignorieren, kann nicht nur Ihr Programm, sondern das gesamte Windows-System abstürzen.

Im Gegensatz zu gewöhnlichen C++-Objekten gibt es bei DirectX keine Konstruktoren und Destruktoren. Für jedes Objekt existiert statt dessen eine Initialisierungs-Funktion. Zusätzlich besitzt jedes DirectX-Objekt eine Release-Funktion als Ersatz für einen Objektdestruktor. Diese Funktionen müssen Sie selbst aufrufen.

Von Version zu Version hat Microsoft einige Änderungen und Verbesserungen an den DirectX-Objekten vorgenommen. Um zu alten Programmen und verschiedenen installierten Versionen von DirectX kompatibel zu bleiben, gibt es einen interessanten Versions-Mechanismus: Wenn Sie von DirectX ein Objekt anfordern, bekommen Sie zunächst ein Objekt der Version 1.0, das Sie nach einer neueren Version von DirectX fragen können.

Eine zentrale Rolle spielen die sogenannten GUIDs (Globally Unique Identifiers). Das sind eindeutige Zahlencodes, die Windows jedem Objekt zuordnet. So kann Windows die Objekte voneinander unterscheiden. Wenn Sie ein DirectDraw-Objekt der Version 3.0 wünschen, sollten Sie die entsprechende GUID für dieses Objekt kennen.

Laut Microsoft soll der zugrundeliegende Algorithmus erst um das Jahr 3400 herum bereits verwendete Identifikationsnummern doppelt vergeben. Diese Weitsichtigkeit erspart der Computerwelt ein ähnliches Chaos wie beim Jahr-2000-Problem.

DirectDraw und dessen Objekte

In den abgedruckten Listingzeilen haben wir der Übersichtlichkeit zuliebe auf die Fehlerbehandlung verzichtet. Dieser Code soll Ihnen das Prinzip und die Schnittstelle nahebringen; guten Programmierstil bietet dagegen der Code der neuen Demobibliothek auf der Heft-CD.

Den Zugriff auf eine Grafikkarte liefert Ihnen das Objekt *IDirectDraw*:

THE PARTY: EIN LEBEN NACH WEIHNACHTEN

Echte Computerfreaks warten nicht aufs Christkind, sondern auf die Zeit danach: Bereits zum achten Mal öffnete *The Party* am 27. Dezember ihre Pforten. Diese Demo-Party, ein Treffpunkt von Demo-Programmierern, Musikern und Grafikern, zählt zu den größten Ereignissen dieser Art in Europa.

Drei Tage lang bevölkerten über 2500 – vorwiegend männliche – Besucher das Messezentrum der kleinen dänischen Stadt Aars an der E45 zwischen Aalborg und Viborg im Norden Jütlands. *Do You Believe in Life after Christmas?* lautete das diesjährige Motto, angelehnt an den Refrain des 98er-Comeback-Hits der Popdiva Cher.

ERSTER PLATZ bei den PC-Demos: *Moai* von Nomad

Traditionell traf sich hier nicht nur die PC-Szene, auch C64-Freaks und die Amiga-Gemeinde waren mit neuen Demos und Effekten vor Ort. Fast alle Größen der europäischen Demo-Szene waren gekommen, um mit Gleichgesinnten Wissen und Quellcodes auszutauschen und auf zahlreichen Wettbewerben ihr Können unter Beweis zu stellen.

Höhepunkte der Party waren wie immer die Präsentationen der verschiedenen Wettbewerbsbeiträge. Die Werke wurden auf einer 10 x 8 Meter großen Leinwand – unterstützt durch eine leistungsstarke Soundanlage – vorgeführt und später vom Publikum bewertet.

Bei den PC-Demos, gewissermaßen der Königsdisziplin, konnten vor allem zwei Programme Aufsehen erregen:

- *Moai* von der Gruppe Nomad wurde zum Sieger gekürt. Es besteht ausschließlich aus 3D-Szenen und zeigt einen antiken Tempel, der sich in eine Disco verwandelt.
- Das zweitplazierte *State of Mind* geht einen anderen Weg: Statt einer Demo mit eigener Musik schuf die Formation Bomb ein neues Musikvideo zu einem bereits vorhandenen Song.

Zu beiden Demos gibt es eine MS-DOS- und eine Windows-Version. *State of Mind* wartet sogar noch mit einer Linux-Variante auf: Ein deutlicher Beweis dafür, daß begnadete Programmierer heutzutage auf jeder Plattform Grafik-Demos schreiben können.

Die Programme, Grafiken und Musikdateien der Wettbewerbe sowie allgemeine Informationen über *The Party* finden Sie online unter

`www.theparty.dk`

Zusätzlich zu den üblichen Wettbewerbskategorien – wie Demo, 64K-Intro (Demo mit einer maximalen Datengröße von 64 KByte), Grafik oder Musik – fand zum ersten mal ein Hacking-Contest statt. Dabei ging es darum, in möglichst kurzer Zeit eine Borderware Firewall zu umgehen und als Beweis für das Gelingen eine Datei auf dem „geschützten" Server abzulegen.

VERLÄNGERTE FEIERTAGE auf *The Party 1998* in Dänemark

Im Java-Wettbewerb konnte das Publikum plattformunabhängige Demos bewundern. Trotz der Leistungseinbußen, die die Emulation einer Java-Maschine mit sich bringt, zeigen die Programme in dieser Disziplin inzwischen einige interessante Effekte.

Als kleine Kuriosität wurde sogar eine Demo auf einer Sony-Playstation präsentiert. Doch die Zukunft der Demo-Programmierung dürfte weiterhin in der Welt von PC, Amiga und C64 liegen – zumal es eine teure Entwicklerversion erfordert, um für diese Spielekonsole zu programmieren.

DER GEWINNER des Raytracing-Wettbewerbs: *Suburbs 2100* von 3D Addict

Wer nicht nur die Mauskugel über den Tisch rollen wollte, konnte sein Geschick bei einem Bowling-Turnier unter Beweis stellen. Weitere Abwechslung von der Bildschirmarbeit brachten verschiedene Live-Auftritte von Rave-Musikern.

Das große, eigens für *The Party* installierte Netzwerk lud auch zum Spielen ein: So mancher Spielefreak vergnügte sich beim Halflife-Wettbewerb, und nostalgisch veranlagte Naturen traten im C64-Spiel *International Karate* gegeneinander an.

Möchten Sie selbst einmal eine dieser großen Partys besuchen, brauchen Sie nicht unbedingt nach Dänemark zu reisen. Die nächste große Veranstaltung, die *Mekka & Symposium 2k-1*, findet vom 2. bis 5. April 1999 in Fallingbostel in der Lüneburger Heide statt. Dort gibt es die üblichen Wettbewerbe jeweils für PC, Amiga und C64. Außerdem versuchen die Teilnehmer bei dem für diese Party typischen 32K-Game-Wettbewerb, ein komplettes Spiel – egal welches Genre – mit allen Daten in 32 KByte zu programmieren. Genaue Angaben zum Veranstaltungsort und den Teilnahmebedingungen finden Sie auf der Web-Site

`http://ms.demo.org`

Um möglichst viele „Scener" (also Leute aus der Demo-Szene) anzulocken und gleichzeitig die hauptsächlich an Spielen interessierten Besucher fernzuhalten, soll diesmal das Spielen auf dem Partynetzwerk unterbunden werden. Eigens für Multiplayer-Spiele findet eine Woche später in derselben Halle die sogenannte *Planet Insomnia*-Spiele-Party statt, nähere Infos dazu finden Sie auf der Site

`www.planet-insomnia.de`

Die Siegerdemos *Moai* und *State of Mind* finden Sie auf der Heft-CD.

```
IDirectDraw * dd = NULL;
GUID * ddGUID = NULL;
DirectDrawCreate(ddGUID,&dd,
  NULL);
```

Dieser Code erzeugt ein *IDirectDraw*-Objekt und speichert den Pointer darauf in *dd*. *ddGUID* dient dazu, mehrere im System installierte Grafikkarten zu unterscheiden. Falls Sie – wie hier im Beispiel – *ddGUID* auf 0 setzen, kommt die Standard-Grafikkarte zum Einsatz.

Im nächsten Schritt teilen Sie Windows mit, daß Ihr Programm von nun an der alleinige Besitzer der Grafikkarte sein soll. Dies erreichen Sie mit

```
dd->SetCooperativeLevel(
  ParentWindow,
  DDSCL_EXCLUSIVE |
  DDSCL_FULLSCREEN |
  DDSCL_ALLOWREBOOT);
```

Das erste Argument, das Sie übergeben, ist der Handle eines Fensters. Wie Sie die Fensterklasse definiert haben, ist egal – sie muß allerdings vom aktuell laufenden Programm erzeugt worden sein. Die drei durch ein logisches Oder verknüpften Flags im zweiten Parameter geben Ihnen vollen Zugriff auf die Grafik-Hardware.

Ist obiger Befehl ausgeführt, wirkt sich jeder Absturz fatal auf Windows aus. Sollte Ihr Programm abstürzen, können Sie den Fehlerdialog weder sehen noch bedienen, sondern müssen den Rechner neu starten. Jetzt brauchen Sie eine neuere Version des DirectDraw-Objekts:

```
IDirectDraw2 * dd2 = NULL;
dd->QueryInterface(
  IID_IDirectDraw2,
  (void **) &dd2);
```

IDD_IDirectDraw2 ist die GUID der zweiten Version von DirectDraw. Die Variable *dd2* ist nach Aufruf dieser Funktion ein Objekt vom Typ *DirectDraw2*. Damit können Sie den Videomodus wechseln:

```
dd2->SetDisplayMode(
  320,240,16,0,0);
```

Die ersten drei Parameter stehen für die Breite, Höhe und Farbtiefe des Videomodus. Mit dem vierten Parameter ändern Sie die Bildwiederholfrequenz. Eine 0 setzt die Wiederholfrequenz auf Standardwerte. Das letzte Argument hat noch keine Bedeutung und ist für spätere Erweiterungen von DirectX gedacht.

Die erste Hürde ist genommen: Sie haben einen Videomodus Ihrer Wahl und sind im Exclusive-Modus von DirectDraw. Aber wie schreiben Sie jetzt Daten in den Grafikspeicher? Dafür brauchen Sie weitere Objekte.

DirectDraw-Surfaces

Mit den sogenannten Surfaces (Oberflächen) verwalten Sie den Videospeicher. DirectDraw bietet viele verschiedene Arten von Surfaces. Solange Sie nur an einem einfachen Zugriff auf den Videospeicher interessiert sind, bleibt alles relativ einfach:

```
DDSURFACEDESC SurfaceDesc;
memset(&SurfaceDesc,0,
  sizeof(SurfaceDesc));
SurfaceDesc.dwSize =
  sizeof(SurfaceDesc);
```

Um die gewünschten Eigenschaften festzulegen, füllen Sie eine Struktur vom Typ *DDSURFACEDESC* aus. Machen Sie das sorgfältig, denn (wie bereits er-

DEMOS ALS BILDSCHIRMSCHONER

Sie möchten Ihrem Windows-System eine persönliche Note geben? Dann verwenden Sie Ihre bisher geschriebenen Demos als Bildschirmschoner.

Ein Windows-Bildschirmschoner mit der Dateiendung *.scr* ist vom Aufbau her identisch mit einer *exe*-Datei. Wie sich das jeweilige Programm verhält, entscheidet sich beim Aufruf:

- Mit dem Kommandozeilenparameter */c* starten Sie einen Konfigurationsdialog,
- mit */s* einen Schoner.
- Ein Aufruf ohne Parameter – etwa wenn Sie selbst eine *scr*-Datei ausführen – wird wie ein Start mit dem Argument */s* behandelt.

Um die Programmierung eigener Bildschirmschoner zu erleichtern, stellt Microsoft über die Win32-API die Bibliothek *scrnsave.lib* bereit, die alle Windows-spezifischen Aufgaben erledigt. Sie legt automatisch ein Vollbildfenster an, deaktiviert den Mauszeiger und setzt die registrierte Fensterklasse auf *WS_EX_TOPMOST*, damit sich das Fenster immer im Vordergrund befindet. So können Sie sich ganz auf die Gestaltung der Dialoge und des Bildschirmschoners konzentrieren.

Ebenfalls in dieser Bibliothek befinden sich die Funktion *WinMain* und der Message-Handler. Letzterer steuert alle für Bildschirmschoner typischen Verhaltensweisen: So wird der Bildschirmschoner beim Bewegen der Maus oder durch einen Tastendruck beendet.

In unserer neuen Grafikbibliothek *demosys.cpp* wählen Sie mit

```
#define SCREENSAVER
```

die Kompilierung zum Bildschirmschoner. In diesem Fall werden die von *scrnsave.lib* geforderten Schnittstellen-Prozeduren definiert und die Header-Datei *scrnsave.h* eingebunden. Die Prozedur

```
LONG WINAPI ScreenSaverProc(...)
```

enthält Ihren Message-Handler für den Bildschirmschoner. Alle Nachrichten, die Sie nicht bearbeiten wollen, übergeben Sie an den von *scrnsave.lib* bereitgestellten Message-Handler *DefScreenSaverProc(...)*. Nach dem Empfang der Nachricht *WM_CREATE* initialisieren Sie – wie bisher in *WinMain(...)* – die Demobibliothek und starten den Thread des Demos. Erhalten Sie die Nachricht *WM_DESTROY*, rufen Sie zum Beenden die *demoquit()*-Funktion der Demo auf.

An der Demo selbst ändert sich nichts. Damit können Sie alle bisher in PC Underground entwickelten Programme ohne Änderungen als Bildschirmschoner verwenden.

Bei der Prozedur *ScreenSaverConfigureDialog(...)* handelt es sich auch um einen Message-Handler, der einen Konfigurationsdialog bereitstellen soll. Sie können einen solchen Dialog mit einem Ressourcen-Editor erzeugen und an dieser Stelle in das Demosystem einbauen. Die Konfigurationsdaten sollten Sie in der Registry sichern. Wünschen Sie keinen Konfigurationsdialog, verwenden Sie einfach folgenden minimalen Dialog, der nicht einmal ein Fenster öffnet:

```
BOOL WINAPI
  ScreenSaverConfigureDialog(
➥  HWND hdlg,UINT message,
➥  WPARAM wparam,LPARAM lparam)
{
  switch (message)
  {
    case WM_INITDIALOG:
      return TRUE;
    case WM_COMMAND:
      switch (LOWORD(wparam))
      {
        case IDOK:
        case IDCANCEL:
          EndDialog(hdlg,TRUE);
          return TRUE;
      }
  }
  return FALSE;
}
BOOL WINAPI
  RegisterDialogClasses(
  HANDLE hinst)
{
  return TRUE;
}
```

Ein Bildschirmschoner unter Windows enthält normalerweise alle Daten (Bitmaps, Wave-Dateien oder Videosequenzen) als Ressourcen; das heißt, sie sind in der *exe*- bzw. *scr*-Datei enthalten. Unser Programmbeispiel *PC Underground Screen Saver.scr*, das Sie in das Windows- oder das System32-Verzeichnis kopieren, verlangt die Bitmap-Grafik *tunnel2.bmp* im Hauptverzeichnis Ihrer Festplatte *C:*.

Ihren Bildschirmschoner sollten Sie im DirectX-Vollbildmodus laufen lassen, weil die GDI-Funktion *StretchDIBits(...)* zum Skalieren eines Bildes auf Bildschirmgröße sehr langsam ist.

UNSER BILDSCHIRMSCHONER greift für den Tunnel-Effekt auf diese externe Bitmap-Datei zu.

wähnt) ist DirectDraw nicht gerade fehlertolerant.

Die Struktur *SurfaceDesc* füllen Sie zuerst mit Null-Bytes und initialisieren das Feld *dwSize* mit der Größe der Struktur. DirectDraw stellt damit fest, mit welcher Version von DirectX Sie Ihr Programm übersetzt haben.

```
SurfaceDesc.ddsCaps.dwCaps =
  DDSCAPS_PRIMARYSURFACE |
  DDSCAPS_FLIP |
  DDSCAPS_COMPLEX;
SurfaceDesc.dwBackBufferCount=1;
```

Die Daten im Feld *ddsCaps.dwCaps* beschreiben die Art der Oberfläche, die Sie anfordern: hier darstellbaren Videospeicher (*DDSCAPS_PRIMARYSURFACE*), der Page-Flipping (*DDSCAPS_FLIP* und *DDSCAPS_COMPLEX*) beherrscht. Das heißt: Sie können zwischen mehreren virtuellen Bildschirmen hin- und herschalten. Für das Page-Flipping benötigen Sie mindestens noch eine zweite Bildschirmseite. In *dwBackBufferCount* geben Sie die Anzahl der zusätzlichen Bildschirmseiten an und legen in *dwFlags* fest, daß Sie folgenden Wert setzen wollen:

```
SurfaceDesc.dwFlags =
  DDSD_CAPS |
  DDSD_BACKBUFFERCOUNT;
```

Teilen Sie DirectDraw mit, welche Informationen Sie in der Struktur gesetzt haben. Da viele verschiedene Arten von Surfaces existieren, muß DirectDraw genau wissen, welche Art von Surface Sie haben möchten.

Dieser Code legt die Oberfläche nach Ihren Wünschen an:

```
IDirectDrawSurface ddSurface=0;
dd2->CreateSurface(
  &SurfaceDesc,&ddSurface,0);
```

Dabei wird *ddSurface* – falls Sie keinen Fehler gemacht haben – mit einem *IDirectDrawSurface*-Objekt initialisiert.

Page-Flipping unter DirectDraw

Mit dem in DirectDraw eingebauten Page-Flipping wechseln Sie schnell zwischen mehreren Bildschirmseiten. Das Prinzip ist sehr einfach: Die Oberfläche, die Sie eben angelegt haben, besteht aus zwei Bildschirmseiten. Eine davon ist sichtbar, während Sie den Inhalt der anderen Seite ändern können, ohne Darstellungsfehler zu erhalten. Sie brauchen sich nicht einmal darum zu kümmern, welche der Seiten gerade sichtbar ist. DirectDraw übernimmt diese Verwaltungsaufgabe für Sie.

Wenn Sie herausfinden wollen, welche Oberfläche Sie gerade ändern dürfen, fragen Sie Ihre sichtbare Surface einfach nach dem BackBuffer, also der zweiten Bildschirmseite. Füllen Sie eine *DDSCAPS*-Struktur, und teilen Sie DirectDraw mit, daß Sie am BackBuffer interessiert sind:

```
DDSCAPS caps;
caps.dwCaps=DDSCAPS_BACKBUFFER;
```

Nun fordern Sie von der aktiven Surface die nächste zum Zeichnen verfügbare Seite an:

```
IDirectDrawSurface * dds;
ddSurface->GetAttachedSurface(
  &caps,&dds);
```

Die Variable *dds* wird dabei mit der Hintergrund-Surface initialisiert, und Sie dürfen mit dem Zeichnen anfangen.

Wenn Sie auf eine Surface zugreifen, ändern Sie immer automatisch die nicht sichtbare Bildschirmseite. Sobald Sie DirectDraw mitteilen, daß Sie fertig sind und umschalten möchten, werden die beiden Seiten ausgetauscht.

Das kostet kaum Rechenzeit, da der Wechsel der Bildschirmseiten in der Grafik-Hardware vonstatten geht. Die Surfaces befinden sich – sofern genug Grafikkartenspeicher vorhanden ist – im Speicher der Karte und nicht im Hauptspeicher des Computers. Zudem wartet DirectDraw vor dem Umschalten darauf, daß der Monitor das Bild komplett aufgebaut hat. Diese Vorgehensweise verhindert Darstellungsfehler, die zum Beispiel entstehen, wenn Sie während des Bildaufbaus auf das nächste Bild umschalten.

Page-Flipping mit einem Bild im Hintergrund heißt Double-Buffering. Es funktioniert aber auch mit zwei (Triple-Buffering) oder mehr inaktiven Bildschirmseiten. Verwenden Sie etwa eine Zeichenroutine, die ein Bild schneller aufbaut als der Monitor, können Sie einige Bilder schon im voraus berechnen. Diese werden dann automatisch der Reihe nach abgespielt.

So starten Sie das Page-Flipping bei DirectDraw: Nachdem Sie ein Bild vollständig gezeichnet haben, rufen Sie die *Flip*-Funktion des *IDirectDrawSurface*-Objekts auf:

```
ddSurface->Flip(
  0,DDFLIP_WAIT);
```

Übergeben Sie der Funktion zwei Parameter. Mit dem ersten ändern Sie die automatische Reihenfolge des Page-Flipping. Für unsere Zwecke ist das uninteressant. Der zweite Parameter *DDFLIP_WAIT* signalisiert, daß Sie mit dem Umschalten warten möchten, bis der Monitor das Bild komplett aufgebaut hat. Der Wechsel zwischen den Bildschirmseiten geschieht also genau dann, wenn der Rasterstrahl das untere Ende des Monitors erreicht hat und wieder nach oben an den Anfang läuft.

Zugriff auf das Surface-RAM-Flipping

Eine wichtige Frage ist noch unbeantwortet: Wie greifen Sie auf den Speicher des verdeckten Bildes zu, um dessen Inhalt zu ändern? Das Objekt *IDirectDrawSurface* stellt hierfür zwei Funktionen zur Verfügung: *Lock* und *Unlock*.

DIE KOMPONENTEN DES DIRECTX-SDK

Komponente	Funktion
DirectDraw	Direktzugriff auf Bitmaps im Grafikspeicher, schnelles Hardware-Flipping
DirectSound	Mischen und Wiedergabe von Sounds
DirectPlay	Verbindung für Modem- und Netzwerkspiele
Direct3D	Komplettes 3D-Grafiksystem mit direkter Kontrolle der Rendering-Pipeline
DirectInput	Eingaberoutinen für Joystick, Maus und Tastatur
DirectSetup	Installationsprozedur für DirectX
AutoPlay	Automatischer Programmstart

Erneut kommen Sie nicht daran vorbei, eine DirectDraw-Struktur vom Typ *DDSURFACEDESC* auszufüllen:

```
DDSURFACEDESC
  SurfaceDescription;
memset(&SurfaceDescription,0,
  sizeof(DDSURFACEDESC));
SurfaceDescription.dwSize =
  sizeof(SurfaceDescription);
```

Dann rufen Sie die Funktion *Lock* auf, die Ihnen die Speicheradresse der Grafikdaten verrät:

```
ddSurface->Lock(0,
&SurfaceDescription,
  DDLOCK_SURFACEMEMORYPTR |
  DDLOCK_WAIT,0);
```

Der Pointer *SurfaceDescription.lpSurface* zeigt nun auf das Video-RAM. Auch einige andere Felder der Struktur enthalten wichtige Informationen. So gibt das Feld *SurfaceDescription.lPitch* an, wie viele Bytes Speicher DirectDraw für eine Bildschirmzeile verwendet. Das mutet im ersten Moment etwas ungewöhnlich an, ist aber für viele Grafikkarten erforderlich.

Wenn Sie zum Beispiel einen 320 x 240 Pixel großen Videomodus in Highcolor setzen, belegt eine Grafikzeile genau 640 Byte. Viele Grafikkarten arbeiten jedoch schneller, wenn dieser Wert zwar etwas größer, aber rechnerisch einfacher zu handhaben ist als die mindestens benötigten Bytes pro Zeile. Sie sollten dies beim Schreiben in den Videospeicher unbedingt beachten.

Nach dem Zeichnen rufen Sie die *Unlock*-Funktion auf:

```
ddSurface->Unlock(
  SurfaceDescription.lpSurface);
```

Halten Sie die Zeit zwischen *Lock* und *Unlock* immer so kurz wie möglich. Während Sie auf den Videospeicher zugreifen, bleibt fast das gesamte Betriebssystem stehen. Nur noch Sie bzw. Ihr Programm bekommt Prozessorzeit. Bedenken Sie: Wenn Sie viel Rechenzeit beanspruchen, werden eventuell wichtige Systemprozesse behindert.

Grün bevorzugt

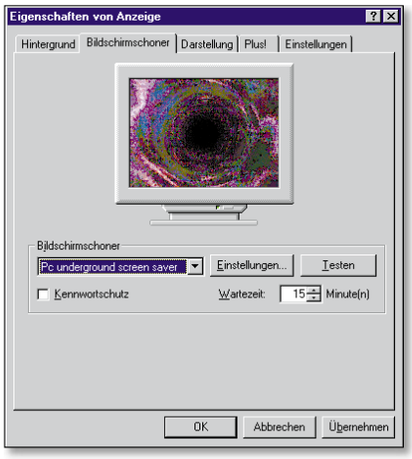
Jetzt sehen wir uns das 16-Bit-Farbmodell genauer an. Ein 16 Bit breites Highcolor-Pixel ist aus drei Feldern aufgebaut: Sie entsprechen den drei Farbkomponenten Rot, Grün und Blau (RGB). In der Regel werden die 16 Bits so aufgeteilt, daß Rot und Blau je 5 Bit bekommen, während Grün mit 6 Bit bevorzugt behandelt wird. Der Grund: Grün ist die Primärfarbe mit der größten Helligkeit, das Auge kann sie am besten wahrnehmen. Im Schema sieht das so aus:

```
RRRRR GGGGGG BBBBB
```

Einige Grafikkarten verwalten die Bits jedoch auf andere Weise. Sie verwenden einheitlich für jede Primärfarbe 5 Bit und lassen dafür das oberste Bit ungenutzt:

```
0 RRRRR GGGGG BBBBB
```

DirectDraw gibt Ihnen auf einigen Grafikkarten diesen 15-Bit-Farbmodus, obwohl Sie einen 16-Bit-Modus setzen wollten. In diesem Fall wandeln Sie die Pixel während des Kopierens in das andere Farbformat um, um zur bisher in PC Undergound verwendeten Grafikbibliothek kompatibel zu bleiben.



DEN AKTIVEN BILDSCHIRMSCHONER legen Sie in der Systemsteuerung im Menüpunkt *Anzeige* fest.

Damit Sie sich künftig nicht mehr darum zu kümmern brauchen, enthält der Code der neuen DirectX-Bibliothek bereits eine effiziente Umwandlungsroutine. Dieser zusätzliche Verwaltungsaufwand ist der Preis für die schnelle Grafik. Das Windows-GDI-Interface würde Ihnen auch diese Arbeit abnehmen.

Die Umwandlung von 16 nach 15 Bit nehmen Sie mit einigen einfachen Operationen parallel für jeweils zwei Pixel vor:

```
unsigned long blau =
  pixel & 0x001f001f;
unsigned long rotgrün =
  (pixel > 1) & 0xffe0ffe0;
pixel = blau | rotgrün;
```

Zuerst maskieren Sie den Blau-Anteil aus, da er sich während der Umwandlung nicht ändert. Die beiden Farbkomponenten Rot und Grün schieben Sie zunächst binär nach rechts und maskieren die gewünschten Bits. Wenn Sie beide Farbanteile mit einer Oder-Verknüpfung wieder zusammenfügen, haben Sie das unterste Grün-Bit weggeworfen und alle übrigen Farbanteile auf die richtige Position geschoben.

Wir haben diese Bibliotheksroutine auch in Assembler programmiert. Sie ist damit nur minimal langsamer als das direkte Kopieren des Speichers.

Wenn Sie bereits mit der bisher in PC Underground verwendeten Grafikbibliothek experimentiert haben, wird Ihnen der Einsatz der neuen DirectX-Erweiterung leichtfallen.

Die einzige auffallende Änderung ist ein neuer Fenstermodus. Neben den vordefinierten Konstanten *FENSTER*, *SKALIERBAR* und *VOLLBILD* für die GDI-Routinen gibt es zusätzlich *DDVOLLBILD* für Vollbilddemos, die die Geschwindigkeit von DirectDraw ausnutzen.

Kompilation ohne Komplikation

Um DirectX-Programme zu kompilieren, benötigen Sie das DirectX-SDK (Software Development Kit) von Microsoft. Sie beziehen es über die Internet-Seite von Microsoft unter

```
http://msdn.microsoft.com/
➥developer/sdk/directx.htm
```

Die Aufgaben seiner Komponenten entnehmen Sie der Tabelle (vorige Seite unten). Aber Achtung: Nicht jede Version von DirectX funktioniert mit jedem Compiler. Benutzer von Microsoft Visual C++ sind hier im Vorteil: Sie brauchen nichts zu tun.

Bei Watcom C++ sieht das etwas anders aus. Mit Version 11 des Compilers erhalten Sie das DirectX SDK 3.0. Damit können Sie die hier entwickelten Programme problemlos kompilieren. Wenn Sie eine neuere Version des SDK installieren, werden Sie einiges an Handarbeit leisten müssen, um alles zum Laufen zu bringen.

Die DirectX-Bibliothek wird nicht – wie die Standardbibliotheken – automatisch zum Programm gelinkt. Darum müssen Sie sich selbst kümmern. Für die Arbeit mit DirectDraw binden Sie die beiden Libraries *ddraw.lib* und *guids.lib* (bzw. *dxguids.lib* bei Visual C) ein.

PEI/BM

Den kompletten Bildschirmschoner, die DirectX-Bibliothek und alle Quelltexte finden Sie auf unserer Heft-CD, Rubrik *Praxis*, und im Internet-Angebot des PC Magazin unter

```
www.pc-magazin.de/magazin/
➥extras.htm
```

Klicken Sie unter *Online Extras* im Menü *Praxis* auf das entsprechende *Download*-Feld.