AUF DER HEFT-CD

Demo-Programmierung unter Windows 95/98/NT

Sternenhagel unter der Lupe

Durch **geschickte Eingriffe** in das Windows-System vergrößern Sie Teile Ihres Desktops und lassen Sterne rieseln.

CARSTEN DACHSBACHER / OLIVER KÄFERSTEIN

In dieser Ausgabe programmieren wir eine Lupe für den Windows-Desktop, die Sie mit der Maus an die näher zu betrachtende Stelle fahren können. Die Schwierigkeit besteht darin, solch ein kleines, frei bewegliches Grafikobjekt (Sprite) so über den Desktop zu bewegen, daß der Hintergrund korrekt dargestellt wird. In einem zweiten Programmierbeispiel zeigen wir Ihnen, wie Sie ins Windows-System eingreifen, um die Bewegungen des Mauszeigers abzufangen.

Sprites auf dem Desktop

In der letzten Ausgabe von PC Underground haben Sie gesehen, wie Sie beliebig geformte Fenster verwenden. Warum sollte man nicht auch animierte Sprites mit dieser Technik realisieren? Auf den ersten Blick wären damit alle Darstellungsprobleme gelöst: Windows restauriert den Hintergrund selbst.

Doch so einfach ist das nicht. Dieser Ansatz funktioniert nur, solange das Fenster seine Form beibehält. Sobald Sie die Konturen ändern, also dem Fenster eine neue Window-Region zuordnen, kann Windows ins Schleudern geraten und mit unschönen Darstellungsfehlern reagieren – abhängig vom jeweiligen Bildschirm- bzw. Grafikkartentreiber. Das Problem liegt bei der Behandlung von Window-Messages und beim Zeichnen des Desktops.

Sie arbeiten nach wie vor mit rechteckigen Fenstern ohne Rahmen. Der Fensterinhalt, also das Bild, das Sie im Rechteck darstellen, besteht aus Teilen des Hintergrunds und dem darzustellenden Sprite. Der Hintergrund ist ein Screenshot des Desktops. Dieses trickreiche Vorgehen ist ähnlich wie bei den Splash-Screens in der vergangenen Ausgabe. Sie verwenden den Desktop als Hintergrundbild in Ihrem Fenster, zeichnen darauf Ihr Sprite – aber eben nur die sichtbaren Teile – und lassen die übrigen Hintergrundpixel stehen.

Einen Screenshot vom Desktop-Bereich unter dem Fenster erzeugen Sie, bevor Sie Ihr Fenster anzeigen lassen. Die Daten behalten Sie als Hintergrundbild. Das funktioniert, wenn Sie Ihr Fenster an einer Stelle auf dem Bildschirm erzeugen und dort belassen.

Schwierig wird es, wenn Sie das Fenster bewegen wollen. Bei einer kleinen Bewegung überschneidet sich die neue Fensterposition höchstwahrscheinlich mit der alten. Wenn Sie einen Screenshot vom Desktop machen, bekommen Sie Ihr eigenes Fenster mit auf das Bild. Genau das wollen Sie aber vermeiden.

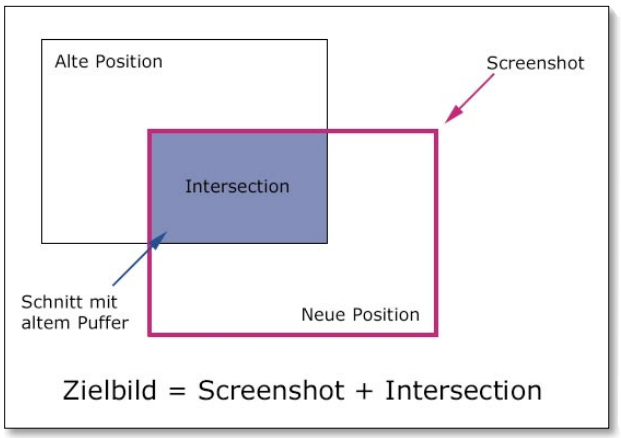
Es gibt mehrere Methoden, um an die ersehnten Hintergrunddaten zu gelangen. Zunächst ein Ansatz, der in eine Sackgasse führt: Sie könnten den Hintergrund in einen eigenen Speicherbereich kopieren und diesen als Fensterhintergrund verwenden. Bei bewegten Fenstern würden Sie dann Ihr eigenes Fenster im Hintergrund sehen, Ihre Sprite-Animation wäre zuerst einmal unsichtbar. Dies funktioniert ohne Schwierigkeiten mit der *WM_HIDE*-Window-Message. Nachdem das Fenster nicht mehr auf dem Desktop zu sehen ist, könnten Sie dann einen Screenshot machen, den Sie mit dem zu zeichnenden Sprite kombinieren und danach im Fenster darstellen.

Das Problem dabei: Alle Änderungswünsche an den Desktop werden in einem Cache verwaltet . Windows nimmt also zuerst einmal alle Zeichenbefehle – unter anderem *WM_HIDE*, *WM_SHOW* und den Befehl für Screenshots – entgegen, führt sie aber nicht zwingend in derselben Reihenfolge und auch nicht zu bestimmten Zeiten aus. Sie können deshalb keine Aussagen über den aktuellen Inhalt oder Zustand des Desktops treffen. Und eine Anfrage an Windows, ob bestimmte Befehle ausgeführt wurden, kommt ebensowenig in Betracht.

Die erste funktionierende Variante nutzt zwei Speicherbereiche (Bitmaps). Diese sind so groß wie Ihr Fenster bzw. Sprite. Die Bitmaps dienen als Hintergrundpuffer. Sie legen sie mit Hilfe der *pcPicture*-Klasse an. Wenn Sie diese Klasse nicht schon aus der letzten Ausgabe kennen, finden Sie den Quellcode und die dazugehörige dokumentierte Header-Datei bei den anderen Quelltexten dieser Ausgabe:

```
//Spriteeffekte und
//Anzeigeroutinen
pcPicture _spr;
//Fensterhintergrund
pcPicture _bkg;
pcPicture _bkg2;

//Sprite laden
_spr.load(„irgendwas.bmp“);
//Speicher reservieren
_bkg.create(_spr.width(),
  _spr.height(), 32, true);
```



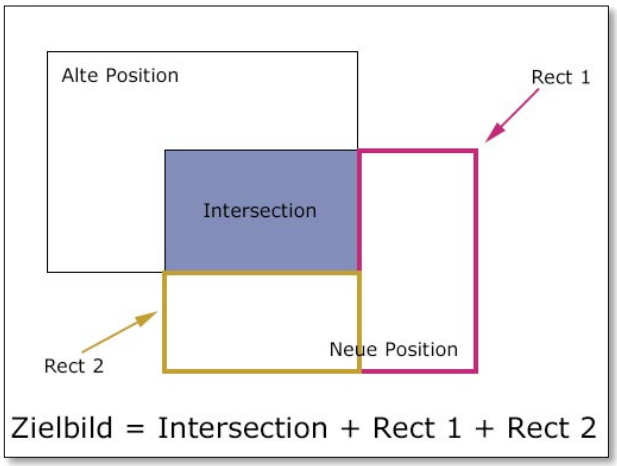
IN DER EINFACHEN VARIANTE der Hintergrundberechnung fallen große Mengen an Screenshot-Daten an.

```
_bkg2.create(_spr.width(),
  _spr.height(), 32, true);
```

Der Wert 32 im Aufruf der *create*-Methode bestimmt, daß Bilder mit einer Farbtiefe von 32 Bit angelegt werden. Dadurch verbraucht Ihr Programm mehr Speicher, vor allem steigt der Kopieraufwand für den Speicherinhalt. Für einen guten visuellen Eindruck ist dieser Aufwand gerechtfertigt: Würden Sie beim Zwischenspeichern nur eine Farbtiefe von 16 Bit einsetzen, käme es bei Desktops mit 32-Bit-Farben zum Verlust von Grafikdaten. Diese würden sich dann in Form von leichten Verfärbungen bemerkbar machen. Umgekehrt können Sie 32-Bit-Screenshots selbst dann verwenden, wenn Ihre Grafikkarte auf 16 Bit eingestellt ist. Windows kümmert sich um die korrekte Farbkonvertierung der Pixel.

Bevor Sie ein Fenster bewegen, fertigen Sie einen Screenshot von dem Bereich an, den das Fenster einnehmen wird. Auf einem Teil dieses Screenshots sehen Sie Ihr eigenes Fenster. Um das zu vermeiden, berechnen Sie, welchen Teil des Bildschirms das Fenster sowohl vor als auch nach der Bewegung überdeckt – der alte Fensterhintergrund ist in einem der Hintergrundpuffer gespeichert.

Wie Sie im Bild auf S. 257 unten erkennen, handelt es sich um die Schnittmenge (das Schnittrechteck). Den Originalinhalt dieses Rechtecks müssen Sie selbst wiederherstellen, weil Sie ihn nicht auslesen können. Sie kopieren den Teil des Hintergrunds, der das Schnittrechteck abdeckt, vom alten in den neuen Hintergrundpuffer.



DIE ELEGANTERE VERSION der Hintergrundberechnung schießt statt einem großen Screenshot zwei kleinere.

Wie das funktioniert, sehen Sie in Listing 1. Bei größeren Sprites dauert das Erzeugen des Screenshots relativ lange, weil dazu Zugriffe auf den Bildschirmspeicher notwendig sind. Deshalb empfiehlt sich diese Methode nur bei kleinen Objekten. Durch den immensen Kopieraufwand bei größeren Fenstern können Sie diese nicht mehr flüssig über den Bildschirm bewegen.

In einer Verfeinerung dieses Verfahrens sparen Sie den Bereich des Hintergrunds, in dem Ihr Fenster liegt, von den Screenshots aus. Wie Sie in der Abbildung S. 258 unten erkennen, berechnen Sie zuerst die Position und Größe der beiden Streifen *Rect 1* und *Rect 2*, die mit neuen Daten vom Screenshot des Desktops gefüllt werden. Wenn Sie nur diese Bereiche kopieren, sparen Sie eine Menge langsamer Zugriffe auf den Bildschirmspeicher. Danach brauchen Sie nur noch das Schnittrechteck von der entsprechenden Position des einen Hintergrundpuffers an die richtige Stelle im anderen zu übertragen.

Durch diese Berechnungen können Sie eine merkliche Geschwindigkeitssteigerung erzielen. Erst durch diesen Kniff können Sie große Sprites ohne Ruckeln auf dem Bildschirm bewegen. Den Code für diese elegantere Version sehen Sie in Listing 2.

Lupenreine Bewegungen

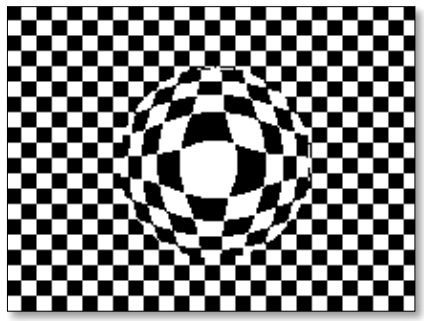
Mit dem beschriebenen Verfahren erzeugen wir eine elektronische Lupe, die Sie über den Bildschirm bewegen können. Auch wenn sich die Form der Lupe nicht ändert – um einen Teil des Bildschirms zu vergößern, brauchen Sie eine Bitmap, die den Bildschirminhalt hinter dem Fenster enthält.

Sie stehen wieder vor dem Problem, daß Sie keinen Screenshot vom Desktop unterhalb Ihres Fensters bekommen. Sie müssen sich das Hintergrundbild wieder selbst zusammenbauen. Das Programm für die Lupe verwendet somit drei Bitmaps:

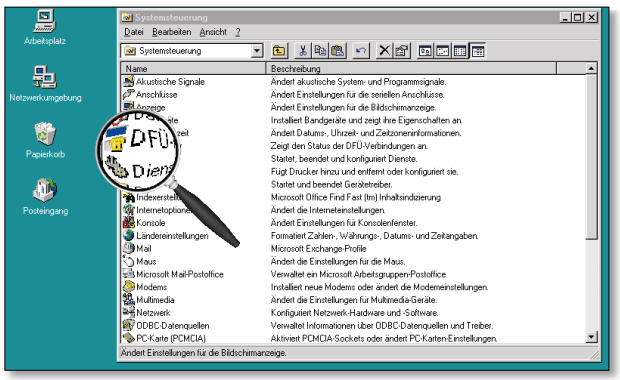
- eine für den Hintergrund,
- eine für die Vergößerung des Hintergrundes
- sowie eine mit dem Bild der Lupe selbst:

```
//Spriteeffekte und
//Anzeigeroutinen
pcPicture _spr;
//Fensterhintergrund
pcPicture _bkg;
pcPicture _bkg2;

//Sprite laden
_spr.load(„lupe.bmp“);
_bkg.create(_spr.width(),
  _spr.height(), 32, true);
_cmb.create(_spr.width(),
  _spr.height(), 32, true);
{...}
//Hintergrund zur Bearbeitung
//in Puffer kopieren
_bkg.blitTO(&wnd->_cmb);
//Lupe berechnen
_lupe.calc(&wnd->_cmb,
  &wnd->_bkg);
//Lupe darüber zeichnen
_spr.blitTRANSTO(&wnd->_cmb);
//... und ab damit ins Fenster
_cmb.blitTO(hdc);
```



DIE LINSE VERGRÖßERT im Zentrum stärker als an den Rändern.



UNTER DER LUPE erkennen Sie auch kleinste Details auf Ihrem Desktop.

Die Funktion *pcLupe.calc(...)* gibt zwei Objekte der *pcPicture*-Klasse zurück. Diese Klasse bietet die Option, einen Zeiger auf die Bitmap-Daten zu bekommen. In diesem Fall handelt es sich um Bitmaps mit 32 Bit Farbtiefe, da Sie die Bitmaps so angelegt haben.

Die Zeiger erhalten Sie mit:

```
pcLupe::calc(pcPicture*
  trgPIC, pcPicture* srcPIC)
{
  {...}
  unsigned int *dst =
    (unsigned int *)trgPIC->
    _bytes;
  unsigned int *src =
    (unsigned int *)srcPIC->
    _bytes;
  {...}
}
```

Jetzt können Sie sowohl auf die Bitmap-Daten des Hintergrundes als auch auf die des Fensters zugreifen. Sie haben fast alles, was Sie zum Zeichnen brauchen: Es fehlt nur noch die Routine für die Darstellung einer Vergrößerungslinse.

Um einen interessanten Verzerrungseffekt wie im Bild links unten zu erreichen, soll die Linse im Zentrum stärker vergrößern als im Randbereich. Dazu lesen Sie jeden Hintergrundpixel, den Sie setzen, an einer leicht verschobenen Position aus. Diese Verschiebung entnehmen Sie einer Tabelle, die Sie während des Programmstarts einmal berechnen.

Um die Tabelle mit Werten zu füllen, gehen Sie wie folgt vor: Zuerst wählen Sie eine bestimmte Breite für die Linse und legen eine Tabelle mit doppelt so vielen Integer-Werten an, wie die Breite zum Quadrat ist, also `Tabellengröße = 2 * Linsenbreite`^{2}

Dann gehen Sie mit zwei Schleifen – die äußere für die Zeilen, die innere für die Spalten – innerhalb der Tabelle jeden Punkt durch. Ermitteln Sie zuerst die Differenz zwischen der Position des aktuellen Punkts und des Mittelpunkts der Linse. Mit diesem „Vektor“ können Sie den Abstand zum Mittelpunkt berechnen.

Wenn Sie diesen Abstand durch den maximalen Abstand vom Mittelpunkt (also durch den Kreisradius, der der halben Breite der Lupe entspricht) teilen, erhalten Sie einen Skalierungsfaktor. Daraus schließen Sie auf die Koordinate des auszulesenden Punktes. Da die Tabelle relative Angaben enthalten soll, ziehen Sie davon noch die Originalposition ab. Was sich in der Beschreibung kompliziert anhört, liest sich im C-Quelltext kurz und bündig:

```
#define LSIZE 100
signed int lupe[LSIZE*LSIZE]
  [2];

int i, j;

for (j=0; j<LSIZE; j++)
  for (i=0; i<LSIZE; i++)
{
  //Vektor bilden
  double x=i-LSIZE/2;
  double y=j-LSIZE/2;

  //Entfernung zum
  //Mittelpunkt der Linse
  double d=sqrt(x*x+y*y);

  //Verhältnis zur maximalen
  //Entfernung
  double q=d/sqrt(
    (LSIZE/2)*(LSIZE/2));

  //Lupe auf runden Rand
  //begrenzen
if (d>(LSIZE/2)) q=1.0;

  //Verschiebung der Koordi-
  //nate des auszulesenden
  //Pixels in Relation zur
  //originalen Position
  double nx=(double)
    (i-LSIZE/2)*(q-1.0);
```

```
    double ny=(double)
      (j-LSIZE/2)*(q-1.0);

    lupe[i+j*LSIZE][0]=(int)nx;
    lupe[i+j*LSIZE][1]=(int)ny;
  }
```

Der folgende Code-Ausschnitt demonstriert, wie einfach der Lupeneffekt ist, wenn Ihnen erst einmal die Tabelle zur Verfügung steht. Um den Bereich der Lupe zu zeichnen, brauchen Sie nur die Zeiger auf die Bitmap-Daten und die Breite der Bitmaps einzugeben:

```
//Zeiger auf Zielbitmap
//und deren Breite
unsigned int *dest;
int dwidth;
//Zeiger auf Quellbitmap
//und deren Breite
unsigned int *source;
int swidth;
//Position der Lupe
int x, y;

for (int j=0; j<LSIZE;
  j++)
  for (int i=0; i<LSIZE;
    i++)
{
  //Koordinaten in Bitmaps
  int nx=i+x;
  int ny=i+y;
  dest[nx+ny*dwidth]=
    source[lupe[i+j*LSIZE]+
    nx+ny*swidth];
  }
```

Diesen Code bauen Sie in die *WM_MOVE*- und *WM_PAINT*-Message-Behandlung des Fensters ein, welches die Lupe darstellen soll. Damit besitzen Sie ein Vergrößerungsglas, das Sie beliebig über den Desktop bewegen können.

Sternchenregen

Jetzt wollen wir Ihrem Windows-Bildschirm noch einen Sternchenregen spendieren. Die einzelnen Sterne sind Instanzen der beschriebenen Fensterklasse für animierte, bewegte Sprites. Es fehlt nur eine geeignete Verwaltung der Sterne.

Dazu legen Sie in Ihrem Hauptprogramm ein Array mit einer festgelegten Anzahl *ANZ_STERNE* von Sternklassen an. Ein Stern enthält Bitmaps, ein Fenster und eine Zeitsteuerung für die Animation. Zudem verfügt er über Funktionen, um sich wieder zu reinitialisieren. Bei dieser Reinitialisierung taucht er an einer zufälligen Stelle des Bildschirms wieder auf und fällt ein Stück herunter.

Ein Stern übernimmt die Kontrolle über seine Animation und seine Bewegung. Im Hauptprogramm befindet sich ein Timer, der das Array mit den Zeigern auf die Sterne im 20-Millisekunden-Takt überprüft und einen Eintrag sucht, in dem sich ein nicht aktiver Stern (dessen Animation abgelaufen ist) befindet. Solche Einträge werden reaktiviert und an einer zufälligen Bildschirmposition neu gestartet.

Sterne funktionieren nur, wenn sie sich bewegen. Nur durch die Bewegung kann ein Sprite die nötigen Daten von Windows erfragen, die für eine korrekte Berechnung des Hintergrunds wichtig sind: Windows kann einem Fenster zwar mitteilen, daß ein anderes Fenster über ihm die Position geändert hat, ohne aufwendige Tricks wissen Sie jedoch nicht, ob ein Fenster unter dem eigenen den Inhalt ändert.

Damit die Sternchen immer in der Nähe des Mauszeigers auftauchen, müssen Sie dessen Positionsänderungen mitbekommen. Dazu installieren Sie einen sogenannten „Hook“ unter Windows. Die Code-Auszüge finden Sie im Unterverzeichnis *MOUSEHOOK* bei den Quelltexten zu dieser Ausgabe.

JEDES DER STERNCHEN steuert sich selbst.

So „haken“ Sie sich in die Mausüberwachung ein:

```
HHOOK SetWindowsHookEx
(
  //Was wollen wir „hooken“
  int idHook,
  //Adresse auf Hook-Prozedur
  HOOKPROC lpfn,
  //Application Instance
  HINSTANCE hMod,
  //Thread ID
  DWORD dwThreadId
);
```

Diese Prozedur verlangt zunächst die Instanz Ihrer Anwendung und des Thread-Identifiers. Außerdem müssen Sie ihr mitteilen, was Sie überwachen wollen – hier die Mausbewegung.

Übergeben Sie als *idHook* die Konstante *WH_MOUSE*. Sie brauchen nur noch einen Zeiger auf eine Hook-Prozedur zu übergeben. Diese führt das System immer dann aus, wenn die Anwendung *GetMessage()* oder *PeekMessage()* aufruft und eine die Maus betreffende Nachricht gesendet wird.

Für diese Hook-Prozedur ist folgende Prozedurdefinition vorgegeben:

```
LRESULT CALLBACK MouseProc
(
  //Hook-Code
  int nCode,
  //Message ID
  WPARAM wParam,
  //Maus-Koordinaten
  LPARAM lParam
);
```

Da sich auch mehrere Programme in die Nachrichtenkette einhaken können, erhält Ihre *MouseProc* von Windows unter Umständen den Auftrag, manche Nachrichten weiterzuleiten. Dies ist immer dann der Fall, wenn der Hook-Code *nCode* kleiner als 0 ist.

Nachrichten leiten Sie mit der folgenden Prozedur weiter:

```
LRESULT CallNextHookEx
(
  //Handle Ihres Hooks
  HHOOK hhk,
  //Hook-Code
  int nCode,
  //Parameter weitergeben
  WPARAM wParam,
  LPARAM lParam
);
```

Der ID-Code im Parameter *wParam* der *MouseProc*-Funktion benennt den Typ der Mausnachricht. Den Wert von *lParam* interpretieren Sie als Zeiger auf eine *MOUSEHOOKSTRUCT*-Struktur:“

```
typedef struct
  tagMOUSEHOOKSTRUCT
{
  //Koordinaten der Maus
  POINT pt;
  //hwnd des Fensters, das
  //die Mousemessage
  //bekommen wird
  HWND hwnd;
  //sonstiges
  UINT wHitTestCode;
  DWORD dwExtraInfo;
} MOUSEHOOKSTRUCT;
```

In dieser Struktur finden Sie die Koordinaten die Sie brauchen, um die kleinen Sternchen wie im Bild unten auf den Desktop loszulassen.

TIP **Die vorgestellten Programme verweigern auf einigen älteren Matrox-Mystique-Grafikkarten ihren Dienst. Das liegt an deren Treibern.** PEI

Die Quelltexte zu den animierten Fenstern finden Sie zusammen mit der zugrundeliegenden Grafikbibliothek auf unserer Heft-CD in der Rubrik *Praxis/Programmierung/PC Underground* und auf unserer Website unter

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

Klicken Sie in der Tabelle *Online Extras* unter *Praxis* auf das entsprechende *Download*-Feld.