Animationen in C

Spiele pr gr ieren

Spiele stellen sich oft durch ein eindrucksvolles Intro vor. Dieser Beitrag zeigt, wie man durchscheindende, bewegte Figuren vor einem feststehenden Hintergrund programmiert.

Peter Winkler

Paletten - Programmierung

Palette – as ist das

Das Bild, das auf einem Bildschirm dargestellt wird, besteht im allgemeinen aus vielen farbigen Punkten, sogenannten Pixeln, die in einer rechteckigen Matrix angeordnet sind.

Im Textmodus können nur ganze Buchstaben, die aus mehreren Pixeln zusammengesetzt sind, verändert werden. Im Graphikmodus kann man jedem dieser Pixeln eine eigene Farbe zuordnen. Nun stellt sich die Frage, wie man dem Computer zum Beispiel eine Mischung zwischen Blau und Grün beibringt. Bei dem PC wurde dieses Problem durch RGB–Anteile gelöst. Jeder Farbe wird somit Rot-, Grün- und Blauanteil zugeordnet.

Eine Palette ist eine Tabelle, in der die RGB–Anteile den einzelnen Farben zugeordnet sind.

Diese Zuordnung wird durch eine Art Zeiger auf einen Paletteneintrag realisiert. Im Bildschirmspeicher steht daher für jeden Pixel eine Zahl, die die Nummer des gewünschten Paletteneintrags angibt und in dem Paletteneintrag stehen die RGB-Werte.

Diese Zuordnung spart einerseits Speicher und andererseits können alle Pixel, die die selbe Farbnummer tragen, auf einmal verändert werden.

Bei hohen Farbtiefen ist diese Vorgangsweise aber nicht mehr sinnvoll und die einzelnen RGB-Werte werden für jeden Pixel einzeln angegeben.

er nderung der Palette

Hier stellt sich die Frage, wie man eine Palette überhaupt verändern kann.

Eine VGA-kompatible Graphikkarte bietet verschiedene Ports an, mit denen man dies realisieren kann.

Pixel Write Adress Port:	Read / Write 3C8h

Auf diesen Port muß die Farbnummer geschrieben werden, bevor die RGB Anteile geschrieben werden können.

Pixel Read Adress Port:	Write only 3C7h

Auf diesen Port muß die Farbnummer geschrieben werden, bevor die RGB Anteile gelesen werden können.

Pixel Color Value Port:	Read / Write 3C9h

Dieser Port stellt den Datenport dar. Hier können die Rot/Grün/Blau-Anteile hintereinander gelesen oder geschrieben werden.

Als erstes möchte ich hier zwei kleine Funktionen vorstellen, die die Anteile einer Farbe setzen oder auslesen.

```
/*********************************************************************
void getcol(int i,unsigned char *r,unsigned char *g,unsigned char *b)
 Ermittelt die RGB - Anteile einer Farbe
 Parameter:
  int i             Farben - Nummer
  unsigned char * r,g,b    Rot / Grün / Blauanteil der Farbe
*********************************************************************/
void getcol(int i,unsigned char *r,unsigned char *g,unsigned char *b)
{
 outp(0x3C7,i);         //Farbnummer bekanntgeben
 *r=inp(0x3c9);         //RGB Werte auslesen
 *g=inp(0x3c9);
 *b=inp(0x3c9);
}
/*********************************************************************
 void setcol(int i,unsigned char r,unsigned char g,unsigned char b)
 Setzt die RGB - Anteile einer Farbe
 Parameter:
  int i             Farben - Nummer
  unsigned char r,g,b    Rot / Grün / Blauanteil der Farbe
*********************************************************************/
void setcol(int i,unsigned char r,unsigned char g,unsigned char b)
{
 outp(0x3c8,i);         //Farbnummer bekanntgeben
 outp(0x3c9,r);         //RGB Werte setzten
 outp(0x3c9,g);
 outp(0x3c9,b);
}
```

Als nächstes möchte ich zwei Funktionen vorstellen, die die gesamte Palette setzten bzw. auslesen

```
/*********************************************************************
 void getpal(unsigned char * palette)
 Ermittelt die Palette, die von der Graphikkarte verwendet wird
 Parameter:
  unsigned char * palette    Erhält RGB Anteile der einzelnen Farben
*********************************************************************/
void getpal(unsigned char * palette)
{
 int i;
 outp(0x3C7,0);         //Auslesen ab Farbe 0
 for(i=0;i<FARBEN_ANZAHL;i++)    //Farben durchlaufen
 {
  palette[i*3]=inp(0x3c9);     //RGB Werte lesen
  palette[i*3+1]=inp(0x3c9);
  palette[i*3+2]=inp(0x3c9);
 }
}
/*********************************************************************
 void setpal(unsigned char * palette)
 Legt die Palette fest, die von der Graphikkarte verwendet werden soll
 Parameter:
  unsigned char * palette    Enthält RGB Anteile der einzelnen Farben
*********************************************************************/
void setpal(unsigned char * palette)
{
 int i;
 outp(0x3c8,0);         //Setzen ab Farbe 0
 for(i=0;i<FARBEN_ANZAHL;i++)    //Farben durchlaufen
 {
  outp(0x3c9,palette[i*3]);    //RGB Werte setzen
  outp(0x3c9,palette[i*3+1]);
  outp(0x3c9,palette[i*3+2]);
 }
}
```

`FARBEN_ANZAHL` Gibt die Anzahl der zu bearbeitenden Farben an.

winklerp@pcc.tgm.ac.at

Mit diesen Funktionen kann man einige schöne Effekte erreichen. So kann man es zum Beispiel bewerkstelligen, daß Wasser am Bildschirm so aussieht, als würde es fließen oder ähnliches, ohne die eigentlichen Bilddaten zu verändern. Dies wird meist durch Rotation eines Teils der Palette realisiert.

Man kann aber auch den gesamten Bildschirm abblenden (abdunkeln) bzw. einblenden (aufdunkeln). Dieser Effekt wird auch als "fading" bezeichnet. Man dekremiert bzw. inkremiert dabei die einzelnen RGB Werte bis der Bildschirm ganz dunkel ist bzw. bis das Bild in seinen Originalfarben dargestellt wird.

Eine Möglichkeit, wie man diesen Effekt bewerkstelligt, zeigen die folgenden Funktionen:

```
/*************************************************************************
 void fadepalout(int from,int count)
 Abdunkeln
 Fadet einen Farb - Block der aktuellen Palette ab (setzt alle Anteile
 auf schwarz)
 Parameter:
  int from           Startfarbe des Blocks
  int count          Anzahl der Farben des Blocks
*************************************************************************/
void fadepalout(int from,int count)
{
 int i,j;              //Zähler
 unsigned char palette[FARBEN_ANZAHL*3];
//Temporär Palette RGB
 getpal(palette);           //Aktuelle Palette ermitteln
 count=(count+from)*3;         //Anzahl der Anteile des Blocks
 for(j=0;j<=FARBEN TIEFE;j++)
 {
  for(i=0;i<count;i++)         //Alle Farbanteile durchlaufen
    if(palette[i]!=0)palette[i]--;
      WaitRetrace();
  setpal(palette);        //Palette setzen
  delay(50);
 }
 return;
}
```

Als erstes wird die aktuelle Palette ausgelesen. Die RGB Werte dieser Palette werden in der Schleife kontinuierlich derkemiert, d.h. die Farben werden immer dunkler. Damit unsere Modifikationen auch sichtbar werden, muß die Palette nach jedem Schleifendurchgang neu gesetzt werden. Das Delay dient nur zur Verlängerung des Vorgangs.

```
/*************************************************************************
 void fadepalin(int from,int count,unsigned char *destpal)
 Aufdunkeln
 Fadet einen Farb - Block der aktuelle Palette auf eine neue Palette
 Parameter:
  int from           Startfarbe des Blocks
  int count          Anzahl der Farben des Blocks
  unsigned char * destpal      Neue RGB - Anteile, auf die
                gefadet werden soll
*************************************************************************/
void fadepalin(int from,int count,unsigned char *destpal)
{
 int i,j;              //Zähler
 unsigned char palette[FARBEN_ANZAHL*3];
//Temporär Palette RGB
 getpal(palette);           //Aktuelle Palette ermitteln
 count=(count+from)*3;         //Anzahl der Anteile des Blocks
 for(j=0;j<=FARBEN TIEFE;j++)
 {
  for(i=0;i<count;i++)         //Alle Farbanteile durchlaufen
  {               //Paletten angleichen
    if(palette[i]<destpal[i])(unsigned char)(palette[i])++;
    else if(palette[i]>destpal[i])palette[i]--;  }
  WaitRetrace();
  setpal(palette);        //Palette setzen
  delay(50);
 }
 return;
}
```

Die Funktion `fadpalin()` ist das Gegenstück zu `fadepalout()`. Die aktuellen RGB-Werte werden dabei einfach der Originalpalette des Bildes (`destpal`) immer mehr angenähert.

Um die Funktion `WaitRetrace()` zu verstehen, muß man wissen, wie der Bildschirm sein Bild aufbaut.

Es wird eine Elektronenkanone benutzt, die Zeile für Zeile vom oberen linken Rand bis zum unteren rechten Rand abfährt und je nach Intensität des Elektronenstrahls ein Aufleuchten eines Pixels verursacht, daß bei ausreichender Geschwindigkeit für unser Auge wie ein konstantes Leuchten aussieht.

`WaitRetrace()` wartet nun unter Verwendung eines VGA–Ports auf den Moment, bei dem die Elektronenkanone am unteren Bildschirmrand angekommen ist, den Elektronenrand abschaltet und zurück auf den oberen linken Rand fährt. Dieses Zurückstellen des Elektronenstrahls wird vertikaler Retrace genannt.

Wird `WaitRetrace()` beim Faden nicht vor dem Setzten der Palette aufgerufen, so kann es zum sogenannten Einschneien kommen. Dabei leuchten einige Pixeln auf, die eigentlich nicht leuchten sollten.

Dies wird dadurch verursacht, daß einige RGB-Werte durch das Programm genau zu dem Zeitpunkt verändert werden, bei dem die VGA-Karte diese zum Monitor schickt, was durch `WaitRetrace()` unterbunden wird.

```
/*************************************************************************
 void
 WaitRetrace(void)
 Wartet auf vertikalen Retrace
*************************************************************************/
void WaitRetrace(void)
{
 while(!(inp(0x3da)&8));
 while(inp(0x3da)&8);
}
```

Zur Demonstration des Effekts kann folgendes Programm dienen:

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
#define FARBEN_ANZAHL 256 //Anzahl der Farben
#define FARBEN TIEFE  64-1  //Höchster Wert, den ein R/G/B Anteil
          //enthalten kann
void getpal(unsigned char * palette)
...
void setpal(unsigned char * palette)
...
WaitRetrace(void)
...
void fadepalout(int from,int count)
...
void fadepalin(int from,int count,unsigned char *destpal)
...
/*************************************************************************/
/***************  H A U P T   P R O G R A M M  ********************/
/*************************************************************************/
void main(void)
{
 int i;              //Zähler
 unsigned char palette[FARBEN_ANZAHL*3];
//Temporär Palette RGB
 randomize();          //Zufallsgenerator aktivieren
 for(i=0;i ;i++)       //Bildschirm füllen
 {
  textcolor(rand()%16);
  cprintf(„Hallo“);
 }
 getpal(palette);           //Aktuelle Palette holen
 fadepalout(0,FARBEN_ANZAHL-1);    //Bildschirm abdunkeln
 delay(500);           //Eine halbe Sekunde warten
 fadepalin(0,FARBEN_ANZAHL-1,palette);
//Farben wieder zurück faden
 return;
}
```

Dieses Demo füllt den Bildschirm mit zufälligen Farben und fadet den Bildschirm dann ab. Nach einer halben Sekunde wird der Bildschirm wieder zurückgefadet und das Programm wird beendet.

Das PCX Graphikformat

Mit Windows und einigen Maustreibern von Mircrosoft wird das Programm Paintbrush ausgeliefert. Dieses Graphikprogramm verwendet ein eigens von der Firma ZSoft entwickeltes Bildformat, das sogenannte PCX-Format. Da Paintbrush weit verbreitet ist, hat sich das PCX-Format in Bereich der pixelorientierten Graphiken als Standard etabliert und wird von einem Großteil aller Graphik – Programme unterstützt.

So wie sich unsere Computerwelt immer mehr verbessert hat, so hat sich auch das PCX-Format immer weiter entwickelt. Es entstanden verschiedene Versionen des PCX-Formats, die Unterschiede beschränken sich aber im wesentlichen nur auf die Farbpalette.

Der PCX Header

Eine PCX Datei besteht - unabhängig von der Version - aus einem 128 Byte langen Header, dem dann die Bilddaten folgen.

Folgende Tabelle gibt den Aufbau des Headers wieder:

Offset	Bytes	Bemerkungen
0	1	Identifikationsbyte: 0Ah = PCX File
1	1	PCX Version: 0 = Version 2.5 2 = Version 2.8 mit Paletten - Information 3 = Version 2.8 ohne Palette - Information 4 = Windows ohne Paletten – Information 5 = Version 3.0
2	1	Flag für komprimierte PCX Files 0 = unkodiert, 1=RLE Kodierung
3	1	Bits per Pixel (bzw. per Plane), Farbtiefe
4	8	Koordinaten des Originalbildes als Worte (Integer) XMIN, YMIN, XMAX, YMAX
12	2	Horizontale Auflösung in dpi (dots per inch)
14	2	Vertikale Auflösung in dpi
16	48	Color Map mit der Definition der Farbpalette. Organisiert als 3*16 Byte Feld
64	1	Reserviert
65	1	Zahl der Farbebenen (maximal 4)
66	2	Bytes pro Bildzeile(gerade Zahl)
68	2	Palette Information: 1=Color , 2=Graustufen
74	58	Leerbytes zum Auffüllen des Headers

Die Zahlen in den Spalten Offset und Bytes sind in dezimaler Schreibweise angegeben.

Das erste Byte dient zur Identifikation, ob es sich überhaupt um ein PXC-File handelt. Wenn dieses Byte nicht dem Wert 0Ah entspricht, handelt es sich nicht um ein PCX-File.

Das nächste Byte gibt die Versionsnummer an, normalerweise sind aber nur PCX-Files mit der Version 3.0 von Interesse.

Das dritte Byte des Headers ist ein Flag, das angibt, ob die Bilddaten komprimiert sind. Eine 1 bedeutet, daß eine Komprimierung vorliegt, bei einer 0 sind die Daten nicht komprimiert (was aber nur sehr selten vorkommt).

Ab Offset 04H beginnt eine Tabelle, bestehend aus vier Worten, die die Dimensionen des Bildfensters in Pixeln angeben. Die Koordinaten werden nach folgender Reihenfolge abgelegt: `XMIN, YMIN, XMAX, YMAX`.

Folglich läßt sich die Größe des Bildes in Pixeln folgendermaßen berechnen:

```
X = XMAX – XMIN + 1
Y = YMAX – YMIN + 1
```

Die Auflösung in dpi ist nur mit Vorsicht zu genießen und im allgemeinen nicht verwendbar.

Die Farbpalette wird in verschiedenen Varianten gespeichert. Bei maximal 16 Farben ist die Palette im Header ab Offset 16 gespeichert. Bei 256 Farben wird die Palette an die Bilddaten angehängt, d.h. sie steht am Ende des PCX-Files. Da Paletten mit 16 Farben heutzutage schon ziemlich mickrig und zusätzlich abhängig von der jeweiligen Graphikkarte sind, werde ich mich nur auf Paletten mit 256 Farben beschränken.

Die Farbdaten sind hier als Tabelle der einzelnen Rot-, Grün-, und Blauanteile (3 Byte pro Farbe!) kodiert. (siehe Abschnitt Palettenprogrammierung)

Ab Offset 66 enthält der Header ein Wort mit der Zahl der Byte pro Bildzeile. Die Zahl bezieht sich auf eine unkomprimierte Zeile, muß aber nicht mit den Pixeln pro Zeile übereinstimmen, da die Bilddaten immer wortweise gespeichert werden.

Am Schluß des Headers finden sich 58 Leerbytes, um die Länge von 128 Byte zu erreichen.

Die Bilddaten

Die Daten werden einfach Zeile für Zeile nacheinander in das File geschrieben. Dabei liegen die Daten aber meist komprimiert vor, da so sonst zu viel Platz benötigt würde. Es wird das sogenannte RLE oder "Run Length Encoding" – Verfahren verwendet. Das Verfahren ist sehr schnell und unkompliziert und hat andererseits eine gute Komprimierungsrate.

Beim RLE Verfahren versucht man einfach, gleiche Pixeln, die nacheinander angeordnet sind, zusammenzufassen und deren Wiederholrate anzugeben.

Man hat folgendes festgelegt:

- Sind die beiden oberen Bits eines Bytes nicht gesetzt (0), dann liegt ein unkomprimiertes Datenbyte vor. Das Byte kann unverändert verwendet werden.
- Sind die beiden oberen Bits eines Bytes gesetzt (1), dann liegt eine komprimierte Information vor. Die Bits 0 bis 5 sind dann als Zähler zu interpretieren, die den Wiederholungsfaktor für das folgenden Byte angeben. Das folgende Byte definiert die Farbe.

Einige Beispiele, um das Verfahren zu verdeutlichen:

```
C3 55          55 55 55
07             07
05             05
C1 F2          F2
```

Bei der Ausgabe auf den Bildschirm muß darauf geachtet werden, daß lediglich der Bereich `XMAX – XMIN` ausgegeben wird, da die Leerbits unterdrückt werden müssen.

Umsetzung in Praxis

Hier möchte ich nun eine Routine vorstellen, die ein PCX-File einliest und entpackt.

Der Funktion wird dabei der Dateiname und zwei Pointer auf Speicherbereiche mitgegeben, die den Header und die Palette aufnehmen sollen. Es wird ein Pointer auf die entpackten Bilddaten zurückgegeben, welcher aber wieder frei gegeben werden muß (siehe Beispiel)!

Sollte die Funktion nicht erfolgreich ausgeführt werden können, wird `NULL` zurückgegeben.

Die Funtkion ist nur für 256 Farben, Version 3–PCX-Files geschrieben und kann (je nach Compiler) nur für Bilder mit einer maximalen Auflösung von 320*200 verwendet werden, da bei größeren Bildern die 64-kByt-eGrenze gesprengt wird und Probleme mit der Speicherverwaltung auftreten.

```
/*********************************************************************/
/* Basic Types. Need these to avoid differences in compilers         */
/* (And C is supposed to be a portable language!)                    */
/*********************************************************************/
typedef signed long int     SLONG;  //32 bit signed.
typedef unsigned long int   ULONG;  //32 bit unsigned.
typedef signed int          SWORD;  //16 bit signed.
typedef unsigned int        UWORD;  //16 bit unsigned.
typedef signed char         BYTE;   //8 bit signed.
typedef unsigned char       UBYTE;  //8 bit unsigned.

#define  TRUE    1
#define  FALSE   0
/*********************************************************************/
/* Pcx Header Struktur                                               */
/*********************************************************************/

typedef struct pcx_header
{
  UBYTE     ID;
  UBYTE     version;
  UBYTE     encoding;
  UBYTE     bits per pixel;
  UWORD     xmin,ymin;
  UWORD     xmax,ymax;
  UWORD     hres;
  UWORD     vres;
  UBYTE     palette16[48];
  UBYTE     reserved;
  UBYTE     color_planes;
  UWORD     bytes per line;
  UWORD     palette_type;
  UBYTE     filler[58];
}pcx_header;


#define PCX BYTEMODE 0           //nächstes Byte bearbeiten
#define PCX_RUNMODE 1            //Wiederholschleife
#define PCX BUFLEN 4*1024        //Bufferlänge beim Einlesen

/*********************************************************************/
/*********************************************************************/
/************       Funktion V loadpcx                  **************/
/*********************************************************************/
/*********************************************************************/

UBYTE * V_loadpcx(UBYTE *filename, struct pcx_header * ph,UBYTE *
palette)
{
  SWORD handle;                     //Dateihandle
  register UBYTE outbyte,bytecount; //Dekomprimierung
  register UBYTE *buffer;           //Pointer für Buffer
  SLONG i;                          //Zähler
  SLONG length;                     //Anzahl der entpackten Bytes
  UBYTE *addr,*startaddr;           //Laufpointer und Startadresse
  SWORD mode=PCX_BYTEMODE;          //Aktueller Modus
  SWORD readlen=0, bufptr=0;        //Buffer - Management


  if(ph==NULL)return NULL;

  /************    Datei öffnen     ************/
  if ((handle = open(filename, O RDWR|O BINARY,S IWRITE)) == -1)
    return NULL;   //Fehler beim öffnen

  /************    Header einlesen  ************/
  read(handle,&ph,sizeof(struct pcx header));

  /************    Header auswerten ************/
  if((ph->ID!=10)||(ph->version=!5)||(ph->bits_per_pixel!=8))
  {
    close(handle);   //Ungültige Daten gefunden
    return NULL;
  }

  /******* Anzahl der entpackten Bytes *********/
  length=((long int)ph->xmax-ph->xmin+1)*((long int)ph->ymax-ph->ymin+1);

  /******* Buffer und Speicher für Bild reservieren *******/
  if((buffer=(UBYTE*)malloc(V_PCX_BUFLEN))==NULL)return NULL;
  if((addr=(UBYTE*)malloc(length))==NULL)
  {
    free(buffer);
    close(handle);
    return NULL;
  }

  startaddr=addr;

  /*************************************************************/
  /*******              Daten entpacken                ********/
  /*************************************************************/
  for (i=0; i<length; i++)
  {
    /*******  nächstes Byte bearbeiten *******/
    if (mode==PCX_BYTEMODE)
    {
      /*******  Buffer wieder auffüllen? *******/
      if (bufptr>=readlen)
      {
        bufptr=0;
        if ((readlen=read(handle,buffer,PCX_BUFLEN))==0)
          break;
      }
      /*******  nächstes Byte holen *******/
      outbyte=buffer[bufptr++];
      /*******    gepackt?       *******/
      if (outbyte>0xbf)
      {
        /*******  Wiederholungszähler berechnen *******/
        bytecount = (int)((int)outbyte & 0x3f);
        /*******  Buffer wieder auffüllen? *******/
        if (bufptr>=readlen)
        {
          bufptr=0;
          if ((readlen=read(handle,buffer,PCX_BUFLEN))==0)
              break;
        }
        /*******  Farbwert holen    *******/
        outbyte=buffer[bufptr++];
        if (--bytecount>0)
          mode=PCX_RUNMODE;
      }
    }
    /*******  Wiederholungszähler ablaufen lassen *******/
    else
      if (--bytecount==0) mode=PCX BYTEMODE;
    /*******  Farbwert kopieren      *******/
    *addr++=outbyte;
  }

  /*******       Palette laden       *******/
  if(palette!=NULL)
  {
    /******* Dateizeiger stellen, einlesen und korrigieren *******/
    lseek(handle,-768L,SEEK_END);
    read(handle,palette,3*256);
    for (i=0;i<256*3;i++)
      palette[i]=palette[i]>>2;  //korrigieren
  }
  close(handle);
  free(buffer);
  return startaddr;
}


/*********************************************************************/
/*********************************************************************/
/************    Hauptprogramm                          **************/
/*********************************************************************/
/*********************************************************************/
void main()
{
  struct pcxheader ph;
  UBYTE palette[3*256];
  UBYTE * picture;
  picture=V loadpcx("Gonzo.pcx",&ph,palette);
....

  free(picture);        //Wichtig!!!
}
```

Wie man die Bilddaten auf den Bildschirm bekommt, zeigt der nächste Abschnitt.

Spiele Programmierung

1. Sprites - rasante Bewegungen auf dem Bildschirm

In einem Rollenspiel als Fantasy-Spielfigur, die über einer Landschaft längst vergangener Zeiten wandert oder als Raumgleiter in einem Aktion-Game : überall werden Sprites benutzt. Im Prinzip kann man so ziemlich alles als Sprite bezeichnen, das sich in irgendeiner Form über einen Hintergrund bewegt.

Nun stellt sich aber die Frage, wie man solche Sprites auf den Bildschirm bringt.

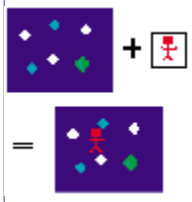
Grundlagen

Das Grundprinzip, um mit Sprites zu arbeiten ist naheliegend: Man kopiert zuerst den Hintergrund auf den Bildschirm und darüber werden dann die Sprites als eine Art Bitmap dargestellt.

Das klingt sehr einfach - ist es aber nicht.

Zuerst kann man sich fragen, ob es sinnvoll ist, immer den gesamten Hintergrund neu darzustellen.

Bevor die Sprites auf ihren neuen Positionen dargestellt werden können, müssen sie zuerst vom Bildschirm gelöscht werden. Nun könnte man immer nur den Bereich des Hintergrundes sichern, der vom Sprite überschrieben wird und diesen dann beim nächsten Bildschirmaufbau über das alte Sprite kopieren und darüber wird dann das neue Sprite dargestellt. Es ist aber meistens schneller, wenn man immer den gesamten Hintergrund darstellt, da bei mehreren Sprites die Verwaltung schwierig und zeitraubend wird.



Als nächstes tritt das Problem auf, daß man Sprites nur als Rechtecke behandeln kann, da alles andere einen zu hohen Rechenaufwand benötigen würde. Nun ist es beinahe unmöglich oder zumindest nicht sehr schön, wenn man alle Sprites als primitive Rechtecke ohne transparente Stellen darstellt. Man könnte so nicht einmal eine einfache Kugel oder ein Dreieck über den Hintergrund darstellen. Es muß also einen Weg gefunden werden, um Bereiche in einem Sprite zu definieren, die transparent (bzw. nicht) dargestellt werden sollen.

Man könnte für jedes Pixel einen Transparenz–Grad definieren oder die durchsichtigen Stellen durch Bitverknüpfungen erreichen. Der einfachste Weg ist aber wohl, daß man eine Farbe als transparent definiert. Beim Darstellen des Sprites werden einfach nur die Pixel gesetzt, die nicht diese Farbe besitzen.

2. Das 2 - Seiten – Prinzip oder Double Buffering

Der Bildschirm baut standardmäßig bei einer Auflösung von 320*200 ein Bild 70 mal in einer Sekunde neu auf.

Jetzt müßte man zwischen zwei dieser Neudarstellungen den Hintergrund und alle Sprites neu bearbeiten. Ansonsten wäre der Zeitraum, in dem der Hintergrund neu dargestellt wird, und alle Sprites überschrieben werden, für den Spieler sichtbar. Dadurch wäre manchmal nur der Hintergrund auf dem Bildschirm zu sehen, und die Sprites würden flimmern und sich mit dem Hintergrund vermischen.

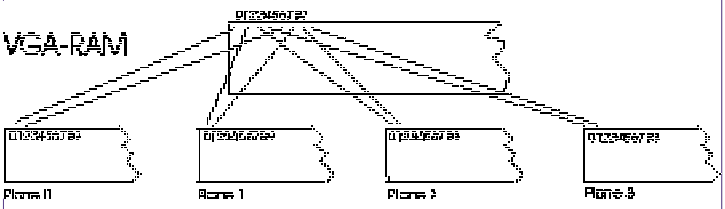
Bei aufwendigeren Spielen ist dieser Zeitraum einfach zu kurz, um die gesamte Bearbeitung des Hintergrundes und der Sprites darin unterzubringen. Darum bedient man sich einer zweiten Bildschirmseite: Eine fertige Bildschirm - Seite wird angezeigt während auf der zweiten Seite das nächste Bild bearbeitet wird; ist die Bearbeitung zu Ende, wird die neue, fertige Seite angezeigt, während auf der anderen wieder das nächste Bild fertiggestellt wird. So hat man immer nur eine fertige Seite auf dem Bildschirm, und ein flimmern wird verhindert.

3. Jetzt wird es ernst : Der Mode X

Eine standardmäßige VGA-Graphikkarte hat mindestens 265K Speicher. Bei einer Auflösung von 320*200 benötigt man 64000 Bytes, um eine gesamte Bildschirmseite zu speichern. Somit kann man theoretisch 4 Bildschirmseiten im VGA-RAM anlegen. Allerdings muß man erst einen Weg finden, um auf die gesamten 256k zuzugreifen zu können, da man über das A000h-Segment (dort wird der VGA-RAM eingeblendet) im Hauptspeicher eigentlich nur auf 64k zugreifen kann. Somit könnte man nur ein Viertel des VGA-RAMs bzw. eine Bildschirmseite verwenden, was es uns unmöglich machen würde, daß Double-Buffering-Prinzip zu verwenden.

Die Lösung bietet hier der sogenannte Mode X. Das Grundprinzip des Mode X liegt darin, das der VGA-RAM auf 4"-Bit-Planes" aufgeteilt wird, die einzeln selektiert und somit in das A000h-Segment eingeblendet werden können.

Wie der Speicher auf die einzelnen Planes aufgeteilt wird, zeigt die folgende Abbildung:



Nun sind nur noch 16 kByte pro Plane für eine Bildschirmseite belegt. Somit kann man noch 3 weiter Bildschirmseiten verwalten, von denen jeweils ein Viertel in die einzelnen Planes eingeblendet wird.

Um eine Plane zu selektieren, muß man über das Indexregister 3C4h auf das Timing-Sequenzer-Register 2 die gewünschte Plane-Nummer schreiben. Dies kann mit einem Word–Zugriff mit einem Befehl erledigt werden.

Timing - Sequenzer (TS) – Register 2	
Bit	Bedeutung
7-4	Reserviert
3	Schreibzugriff auf Plane 3
2	Schreibzugriff auf Plane 2
1	Schreibzugriff auf Plane 1
0	Schreibzugriff auf Plane 0

Beispiel, um Plane 0 zu selektieren:

```
mov dx,03C4h    ;TS - Indexregister
mov al,02h      ;Register 2
mov ah,01       ;Plane 0
out dx,ax       ;selektieren
```

Bevor man aber die einzelnen Planes selektieren kann muß man den Mode X erst einmal initialisieren. Der Mode X basiert auf dem BIOS–Graphikmodus 13h. Allerdings sind zusätzlich einige Manipulationen von VGA–Registern nötig, dessen Erklärung sehr tief in die Thematik eingehen würde und darum den Rahmen dieses Arti-

kels sprengen würde. Darum möchte ich hier einfach nur die Funktion presentieren:

```
void initModeX(void)
{
 asm{
  mov ax,0x0013      //Mode 13h setzen
  int 0x10

  mov dx,0x3c4       //Timing Sequenzer
  mov al,4           //Register 4 (Memory Mode):
  out dx,al          //Bit 3 löschen - Chain4 aus
  inc dx
  in al,dx
  and al,0x0f7
  or al,0x4          //Bit 2 setzen - Odd/Even Mode aus
  out dx,al
  dec dx

  mov ax,0x0f02      //Register 2 (Write Plane Mask):
  out dx,ax          //0fh: alle Planes beim Schreiben ein
  mov ax,0x0a000     //Bildschirmspeicher löschen
  mov es,ax
  xor di,di
  xor ax,ax
  mov cx,0x0ffff
  cld
  rep stosw

  mov dx,0x3d4       //CRTC
  mov al,0x14        //Register 14h  (Underline Row Adress):
  out dx,al
  inc dx
  in al,dx           //Bit 6 löschen - Doubleword adress. aus
  and al,0x0bf
  out dx,al
  dec dx
  mov al,0x17        //Register 17h (CRTC Mode):
  out dx,al          //Bit 6 setzen - Byte Mode ein
  inc dx
  in al,dx
  or al,0x40
  out dx,al
  mov dx,0x03ce      //Write Mode 0 setzen
  mov ax,0x4005      //Über GDC Register 5 (GDC Mode)
  out dx,ax
 }
}
```

Jetzt benötigen wir noch einen Weg, um zwischen zwei oder mehreren Seiten umzuschalten. Das kann man mit den "Linear Starting Address" - Registern durchführen. Mit diesen Registern kann man den Offset festlegenden, bei der die VGA - Karte mit dem auslesen der Bilddaten beginnt. Der Offset errechnet sich mit der Formel off=320 * 200 / 4 * Seitennummer.

Um auf die Register des CRTC (Cathod Ray Tube Controller) zuzugreifen, muß man über das Indexregister 3D4h das gewünschte Register selektieren, um dann das jeweilige CRTC - Register über das Datenregister 3D5h zu bearbeiten. Dies ist wiederum mit einem Wort - Zugriff in einem realisierbar.

CRTC - Register 0Ch: Linear Starting Address High	
Bit	Bedeutung
7-0	Bit 15-8 der 16bittigen Startadresse
CRTC – Register 0Dh: Linear Starting Address Low	
Bit	Bedeutung
7-0	Bit 7-0 der Startadresse

Beispiel um eine neue Startadresse zu setzen. Der benötigte Offset wird aus der Startspalte (x) und der Startzeile (y) berechnent.

```
void setViewStart(SWORD x,SWORD y)
{
 asm{
  mov ax,y
  mov bl,80      //80*4 = 320 Pixel
  mul bl
  add ax,x       //Offset
  mov cl,al
  mov dx,0x3d4   //CRTC
  mov al,0x0c    //Register 0ch(Linear Starting Adress High)
  out dx,ax      //Bits 15:8 setzten
  mov al,0xd     //Register 0dh(LSA Low)
  mov ah,cl      //Bits 7:0 setzen
  out dx,ax
 }
}
```

Und eine Routine, die uns zwischen den ersten zwei Seiten umschaltet:

```
UWORD aktiveViewPage=1;      //Angezeigte Seite
UWORD aktivePage=0xa000;     //bearbeitbare Seite
void switchX(void)
{
 if (aktiveViewPage==0)
 {
  aktiveViewPage=1;
  setViewStart(0,200);       //zweite Seite anzeigen
  aktivePage=0xa000;         //erste Seite bearbeiten
 }
 else
 {
  aktiveViewPage=0;
  setViewStart(0,0);         //erste Seite anzeigen
  aktivePage=0xa000+0x03E8;  //zweite Seite bearbeiten
 }
}
```

Die Variable `aktiveViewPage` dient uns als Erkennungsmittel, welche Seite gerade angezeigt wird. Der Offset ins VGA – RAM der Seite wird gleich mit dem VGA – Segment kombiniert und in `aktivePage` gespeichert.

Als nächstes eine kleine Routine die einen Pixel mit der Farbe col an die Position x,y setzt.

```
void putPixel(SWORD x, SWORD y, BYTE col)
{
 asm{
  mov ax, aktivePage    //Segment laden
  mov es, ax
  mov cx, x             //Plane errechnen
  and cx, 3
  mov ax, 1             //umformen
  shl ax, cl
  mov ah, al
  mov dx, 0x3c4
  mov al, 2
  out dx, ax            //und setzten
  mov ax, 80            //Offset errechnen
  mul y
  mov di, ax
  mov ax, x
  shr ax, 2
  add di, ax
  mov al, byte ptr col
  mov es:[di], al       //Farbe setzten
 }
}
```

3. Das Finale: ein kleines Spiel

Nun wollen wir ein kleines "Spiel" zusammenstellen, daß ein Sprite als unseren Helden über einem Hintergrundbild hin und her pendeln läßt.

Zunächst benötigen wir einen Weg, um unseren Helden und seinem Hintergrund graphisch in unser Spiel zu bekommen. Dazu werden wir die Funktion aus dem Artikel "Das PCX Format verwenden. Die Daten werden wir mit globalen Pointern und die Dimensionen mit jeweils zwei integer – Variablen sichern.

Der Grundkopf mit einigen zusätzlichen Funktionen, die wir verwenden, könnte so aussehen:

```
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
#include <string.h>
#include <sys\stat.h>
#include <alloc.h>

typedef signed long int   SLONG; //32 bit signed.
typedef unsigned long int ULONG; //32 bit unsigned.
typedef signed int        SWORD; //16 bit signed.
typedef unsigned int      UWORD; //16 bit unsigned.
typedef signed char       BYTE;    //8 bit signed.
typedef unsigned char     UBYTE; //8 bit unsigned.

#define  TRUE     1
#define  FALSE    0
/*************************************************************************/
/* Pcx Header Struktur                                                   */
/*************************************************************************/

typedef struct pcx header
{
  ...
}pcx_header;


#define PCX_BYTEMODE 0       //nächstes Byte bearbeiten
#define PCX_RUNMODE 1        //Wiederholschleife
#define PCX_BUFLEN 4*1024    //Bufferlänge beim Einlesen

UWORD aktiveViewPage=1;      //angezeigte Seite
UWORD aktivePage=0xa000;     //bearbeitbare Seite

UBYTE * background=NULL;     //Pointer auf Hintergrund Daten
UBYTE * sprite=NULL;         //Pointer auf Sprite Daten

UWORD backx=0,backy=0;       //Breite / Höhe vom Hintergrund
UWORD spritex=0,spritey=0;   //und vom Sprite
UBYTE palette[256*3];        //Palette des Hintergrunds/Sprites



void initModeX(void);
void setViewStart(SWORD x,SWORD y);
void switchX(void);
void putPixel(SWORD x, SWORD y, BYTE col);

UBYTE * loadpcx
  (BYTE *filename, struct pcx header * ph,UBYTE * palette);
void WaitRetrace(void);
void setpal(unsigned char * palette);
void getpal(unsigned char * palette);
void fadepalin(int from,int count,UBYTE *pal);
void fadepalout(int from,int count);
...

void closemodex(void)   //Schaltet zurück in den Textmodus
{
 asm{
  mov ax,0x3           //Biosinterrupt, Textmodus
  int 0x10
 }
}
```

Jetzt benötigen wir noch zwei Funktionen, die uns unsere Daten auf den Bildschirm bringen. Beim Mode X gibt es da einige Möglichkeiten, ich möchte hier aber nur den wahrscheinlich einfachsten Weg über die Funktion putPixel wählen. Für ein umfangreiches Spiel ist diese Lösung aber nicht verwendbar, da sie viel zu langsam ist.

Eine sehr einfache Möglichkeit um ein Image mit der Höhe höhe und der Breite breite ab der Position x,y darzustellen:

```
void viewImg(UWORD x, UWORD y, UWORD breite, UWORD hoehe,UBYTE *data)
{
 UWORD i,j;
 for(i=y;i<hoehe+y;i++)
 {
  for(j=x;j<breite+x;j++,data++)putPixel(j,i,*data);
 }
}
```

Und das ganze noch einmal für ein Sprite. Als transparente Farbe wurde 0 gewählt.

```
void viewSprite
 (UWORD x, UWORD y, UWORD breite, UWORD hoehe,UBYTE *data)
{
 UWORD i,j;
 for(i=y;i<hoehe+y;i++)
 {
  for(j=x;j<breite+x;j++,data++)if(*data!=0)putPixel(j,i,*data);
 }
}
```

Jetzt fehlt uns nur noch die Hauptfunktion.

Sie muß einerseits unsere Daten laden, und andererseits in einer Schleife unseren Helden hin und herlaufen lassen.

Bei dem Hintergrundbild und dem Sprite ist darauf zu achten, daß diese die selbe Palette haben, da es sonst zu Farbverfälschungen kommt.

```
SWORD main()
{
 struct pcx header head;     //PCX header
 SWORD x=0,inc=1;            //X Position / Richtung des Sprites
 UBYTE tmpPalette[768];      //Temporär - Palette, um den Bilschirm
                             //auf schwarz zu setzten

 printf("Loading ...");

                             //Hintergrund laden
 background=loadpcx("back.pcx",&head,palette);
 if(background==NULL)     //Nicht erfolgreich?
 {//Fehler!!!
  printf("Fehler beim Laden des Hintergrunds");
  return -1;
 }
 backx=head.xmax-head.xmin+1;    // Alles OK à Dimensionen berechnen
 backy=head.ymax-head.ymin+1;

 //Sprite laden
 sprite=loadpcx("sprite.pcx",&head,palette);
 if(sprite==NULL)         //Nicht erfolgreich?
 {//Fehler
  free(background);       //Hintergrund freigeben
  printf("Fehler beim Laden des Sprites");
  return -1;
 }
 spritex=head.xmax-head.xmin+1;  //Alles OK à Dimensionen berrechnen
 spritey=head.ymax-head.ymin+1;
 fadepalout(0,256);              //Vom Textmodus ausfaden
 initModeX();                    //Mode X aktivieren
 memset(tmpPalette,0,256*3);     //Palette auf 0 (schwarz) setzen
 setpal(tmpPalette);
 //Bild auf nicht sichtbarer Seite anzeigen
 viewImg(0,0,backx,backy,background);
 switchX();                      //Seiten umschalten(noch immer
 //unsichtbar, weil Palette schwarz!)
 fadepalin(0,256,palette);       //Palette einfaden
 while(!kbhit())                 //Hauptschleife! Bis Taste
 {                               //Daten anzeigen
  viewImg(0,0,backx,backy,background);
  viewSprite(x,70,spritex,spritey,sprite);
  switchX();            //Seiten umschalten
  if(inc>0)             //Sprite nach rechts bewegen
  {
    if(x<320-inc-spritex)x+=inc; //Geht's noch?
    else inc=-inc;      //Nein! Andere Richtung
  }
  else                  //Sprite nach links bewegen
  {
    if(x+inc>=0)x+=inc; //Geht's noch?
    else inc=-inc;      //nein à Andere Richtung
  }
 }
 fadepalout(0,256);     //Palette Ausfaden
 closemodex();          //In den Textmodus zurück
 free(background);      //Daten freigeben
 free(sprite);
 return 0;
}
```

Ich hoffe, daß ich einigen geholfen habe und wünsche viel Spaß beim Programmieren.