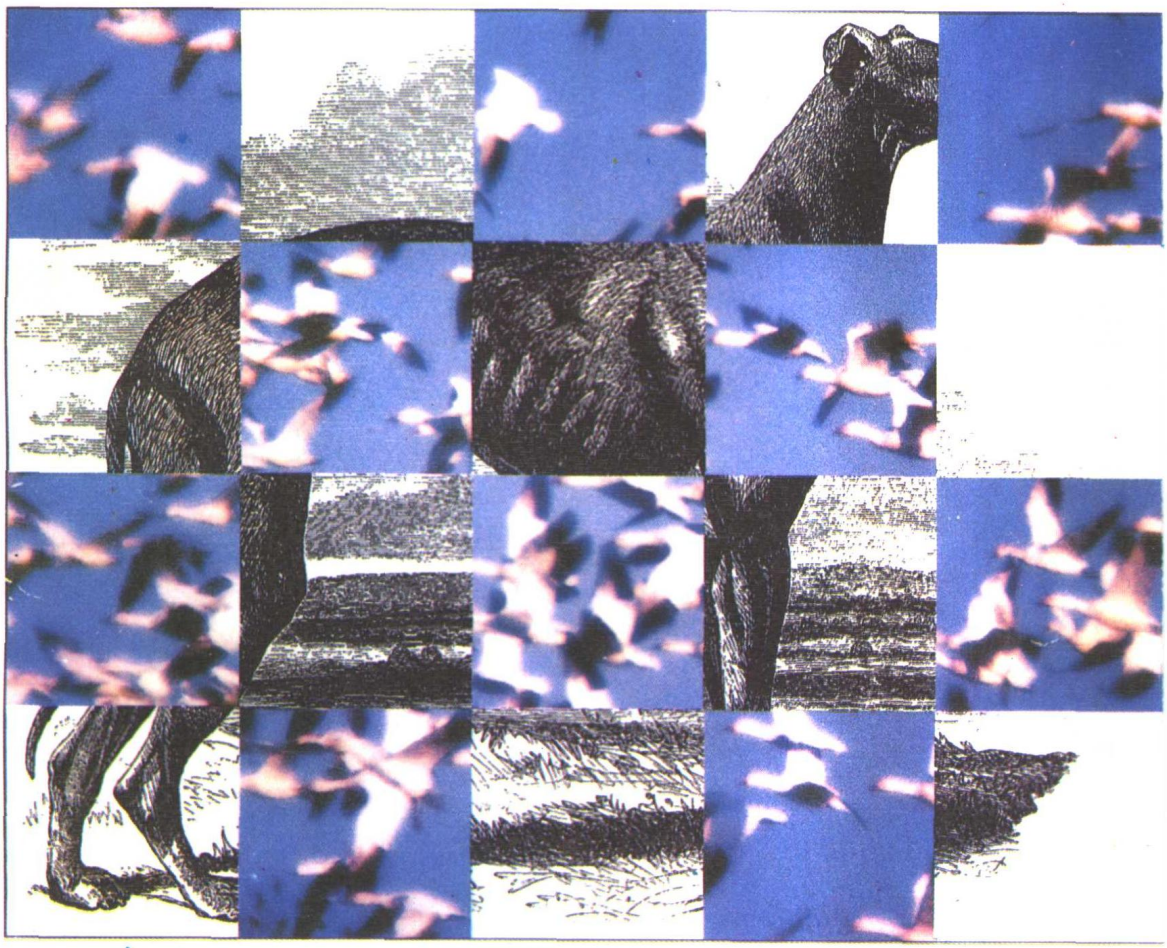
微机图形图像与CAD系列丛书

快速动画制作技术与实例



LEN DORFMAN

- Secrets to smooth-as-glass, high-speed special effects
- Sprite movement & collision detection
- Disk includes a working sprite editor and bit-plane graphics library worth over $200

McGraw-Hill

希望

McGRAW-HILL
学苑出版社

微机图形图像与CAD系列丛书

Lightning-Fast Animation Graphics

快速动画制作技术与实例

Len Dorfman 著
李 鑫 任 英 译
燕卫华 审校

学苑出版社

(京)新登字 151 号

内 容 提 要

本书着重向读者介绍一种易于学到的高速位平面动画制作技术。它以C语言和汇编语言提供一组支持动画制作的基本函数，包括三类图形对象(小精灵、导弹与子弹)的定义、图形对象在位平面内的平滑移动以及相互间的碰撞测试等。这些函数及其可执行程序既可以向读者提供直观的制作结果，同时也可以成为读者制作攻击型视频游戏和特殊的图形动画效果的强有力的支持工具，并可直接应用到读者正在开发的动画程序中。本书既适合动画制作的初学者，也适合于有一定动画制作基础的学习者。

需要本书的读者，请与北京海淀 8721 信箱书刊部联系，邮政编码：100080，**电话：**2562329。

版 权 声 明

微机图形图像与 CAD 系列丛书

快速动画制作技术与实例

著　　者：Len Dorfman
译　　者：李　鑫　任　英
审　　校：燕卫华
责任编辑：甄国宪
出版发行：学苑出版社　　**邮政编码：**100036
社　　址：北京市海淀区万寿路西街 11 号
印　　刷：兰空印刷厂
开　　本：787×1092　1/16
印　　张：23.125　　**字　　数：**539 千字
印　　数：1～5000 册
版　　次：1995 年 11 月北京第 1 版第 1 次
ISBN7-5077-0802-0/TP・13
本册定价：29.80 元

学苑版图书印、装错误可随时退换

致　谢

感谢我的恩师 Barbara、Rachel、Baxter、Aaron、John De F.、Tom H.、John H.、Estelle、Esin、Pierre、Meni、Marc、Ram D.、Ken 和 Treya W.、Sogyal R. 和 Emmanuel，没有他们的教诲与帮助，也就不可能有本书的问世。

作者简介

自从 1973 以来，Len Dorfman 就一直是 New york，Long Island 学校系统的一名教育工作者。由于他对 C 和汇编语言所拥有的广泛的编程经验，他曾是多本书的作者或合著者，如《OS/2 Extra! KBD, MOU and VIO Special Functions Revealed》(McGraw-Hill, 1993)，《Instant OS/2: Porting C Applications to OS/2》(McGraw-Hill, 1993)，《Effective OS/2 Multithreading》(McGraw-Hill, 1994) 和《C Memory Management Techniques》(McGraw-Hill, 1993)。他从 Hofstra 大学拿到教育研究博士学位。

前　言

无论从哪个角度考虑，位平面动画都是一件十分费时的游戏。

必须申明，本书的重点是使C语言应用程序设计者易于学到高速位平面动画制作技术。这一技术通过呈现给应用程序员的C语言图形对象函数库而得以实现。本书支持三种类型的图形对象的开发，这三类对象为“小精灵(sprite)”(在本书中为47×46多色用户自定义图形对象)、“导弹(missile)”和“子弹(bullet)”。

本书提供一些基本函数，这些函数可初始化图形对象，在位平面内平滑移动这些对象，并可测试小精灵、导弹和子弹间的碰撞。这些函数等于向用户提供了若干个工具，这些工具对创建攻击型视频游戏和特殊的图形效果将是十分有用的。

通过包含在本书及磁盘(另售)中的小精灵编辑器程序，可使小精灵对象的生成得以大大简化。此编辑器允许用户使用鼠标来设计多色小精灵形状，然后允许用户将小精灵的形状重新存到磁盘上。通过小精灵编辑器写到磁盘上的小精灵定义文件以易于理解的ANSI标准C格式存储。

第Ⅰ部分是为应用程序员设计的。开始介绍在640×480、16色VGA图形环境中小精灵、导弹和子弹的概念；接着是大量示范程序，这些程序详细说明了有关拖动技术中计算机如何控制小精灵、导弹和子弹的运动；然后给出一种通过键盘控制小精灵运动的方法；其后是测试小精灵、导弹和子弹的碰撞；最后提供正在编制的迷宫中追逐类游戏的源程序代码。

第Ⅱ部分给出了小精灵、导弹和子弹管理程序的源码。特别注意用C语言设计这些程序，以便那些不是汇编奇才的用户也能明白这些代码是如何工作的，且能很方便地修改其源代码。作者未选取这些程序的汇编版，因为在测试中，并未发现汇编和C生成的库模块在动画制作行为方面存在任何真实世界的差别。

第Ⅲ部分给出了小精灵对象编辑器的源码，这一编辑器极大地简化了小精灵的设计。为了设计多色小精灵，只需简单地装入此程序，使用光标定义小精灵对象，然后将此小精灵对象存入磁盘中。小精灵对象以C源码格式存储，此C源码文件可以包含在用户的程序中，并被编译或加到一小精灵对象库中。

注意，位平面动画制作是在PC机中产生动画的众多不同方法中的一种。本书并不是一本动画制作参考手册，而将重点放在易于使高速运动的小精灵、导弹和子弹能包容到用户的程序化“兵器库”中。

使用本书所需的支持环境

用户需要有能支持VGA 640×480、16色视频模式的计算机。在本书中，作者使用了Borland C/C++ 3.1版编译器及TASM汇编器。作者给出了用Microsoft C/C++ 7.0编制的一个示范程序。此C源码易于操作且应易于用其它C编译器编译。

然而，注意作者已将位平面动画制作程序与Borland的图形接口(BGI)集成起来，所以如果用户使用另一编译器，必须使用此编译器的图形接口。但是小精灵代码仍按原样运行，一旦图形模式设置为VGA 640×480、16色模式，此小精灵代码即可运行。

目　录

第 1 部分　位平面动画制作基本元素的管理

第Ⅱ部分　位平面动画制作库的源码

第Ⅲ部分　小精灵编辑器

第Ⅰ部分

位平面动画制作基本元素的管理

本部分主要介绍小精灵、导弹和子弹的基本概念及其在VGA图形中的应用。此部分除一系列逐渐增加其复杂度的编程实例外，还包括其小精灵管理的原型函数。第二章至第八章按其逻辑步骤给出这些实例，各章贯穿用来说明代码的解释性文本，同时还介绍了管理小精灵、导弹和子弹运动的有关技术。第八章的源码将这些程序集成为一个简单拱廊风格式的位平面动画游戏。

第一章　小精灵、导弹和子弹对象

早在八十年代初，作者就购买了一台 Atari 800 家用计算机，以便用于写作第二部小说。此计算机价值 1400 美元，内存为 64K。除了计算机外，作者还买了 AtariWriter 文本处理器磁带及为可爱的女儿购买了 Pac Man 磁带，当然，也不完全如此！

作者插入 Pac Man 磁带并开始玩啊玩。Atari 800 为一杀手游戏机，当费寝忘食地玩了两周后，我向自己提出这样一个问题："啊，我很想知道你是怎样设计出计算机游戏的？"这一问题提出不久就永远地改变了我的生活。

我回到购买 Atari 800 计算机的商店，寻问柜台后面的售货员是否有任何关于如何编制视频游戏的书。得到的回答是："如果你想编制游戏，必须有宏汇编器、一本关于 6502 汇编语言编程的书及一本关于 Atari 硬件的书"，所有这些在本商店中都有。当我胳膊下夹着这些材料并以负债二百多美元的境地离开此商店时，我心情既激动又担心。

我开始读有关 Players、Missiles、磁撞测试记存器以及声音等。我决定写一视频游戏，且三个月以后有一家小型 Atari 软件出版商同意将我的第一份视频游戏 Karmic Caverns 投放市场；然后为 Atari 800 写了两个更具有拱廊风格式的游戏；再接着一些分页作文软件；然后被迫转到 IBM AT 级的计算机上，因为 Atari 软件市场有限。

什么？没有小精灵！仅有来自可怜虫发出的微弱的哗哗声和打嗝声。没有导弹？没有碰撞？我把自己放在怎样一种编程世界中？当然，PCs 机成长起来了，我现在所写的此书是在 33MHz 的 486 微机上运行的，其操作系统可以是 DOS 或 OS/2。

尽管我的兴趣已从热衷于游戏编程转向其他方面，但是，我可以从远处来间接地观察视频游戏的演变。大约两年前，作为一种平时的娱乐，我决定弄弄清楚是否可以将基于 Atari 图形控制器芯片硬件上的大多数功能复制到 PC 机上。我真的这样做了，且本书即是我试图将 32 位、16M、33MHz 的 Local Bus 486 OS/2 2.1 商用级的机器变成 8 位、64K、Atari 800 家用计算机的归档说明。

1.1　什么是位平面

16 色、640×480 VGA 视频模式的物理地址空间是从段地址 0xA000 开始的。屏幕上一个字节为 8 位，各位表示屏幕上的一个像素（PEL、图像元素或点）。640×480 VGA 卡的屏幕，其物理尺寸为 38400 个字节。理由如下：

首先计算一下屏幕上一行有多少个字节，如果在 640×480 模式下屏幕上一行有 640 个像素，640 被 8 除，你可以算出其字节数，应是 80 个字节，所以在 640×480 的屏幕上，每行有 80 个字节。而在 640×480 VGA 屏幕上有 640 列、480 行，只需每行的字节数 80 乘以总行数 480，即可算出 VGA 物理空间的大小，结果为 38400。

从段 0xA000 开始一直延续到 38400 个字节的物理内存是对屏幕的二进制描述。如果地址 A000:0000 对应的值为 0xFF（所有位都为 On），那么屏幕上左上角的第一组 8 个像素

全为开(On)；如果地址 A000:0000 对应的值为 0xAA(每隔一位为 On)，那么左上角的第一组 8 个像素开/关状态为一种交叉状：

地址	Hex	位	像素
A000:0000	0xAA	10101010	On-Off-On-Off-On-Off-On-Off

现在，我希望你会说："等一会儿，如果一物理屏幕位等于 1 时，那么此位对应的像素为 On；如果物理屏幕位等于 0，那么此像素为 Off。位 On、位 Off 为位的两种状态，那么两种位状态如何转换成 16 种颜色？

答案就在 VGA 的结构及图形控制器芯片中。在 640×480、16 色模式中，VGA 图形控制器芯片将每四个位平面(或内存的四个块)映射到 0xA000 屏幕段地址域中。如果内存中有 4 个块映射到物理屏幕区，那么每个像素有 4 个颜色位与之相伴。

映射的平面数	可用的位	二进制视图	十进制视图
4	4	0000 到 1111	0 到 15

0 到 15 的十进制范围参考到有 16 种可能的数字设置，这就是得到 16 种颜色的原因。5 个位平面将参考 32 种颜色；6 个位平面将参考 64 种颜色。

依此类推，你想得到 256 种颜色，需要多少个位平面？如果 4 个位平面(4 个位)产生 16 种颜色，那么 8 个位平面(8 个字节)将可产生 256 种颜色。要显示 1024 种颜色，需要多少个位平面？如果你明白答案为什么是 10 个位平面，那么你已经理解这一内容了。

所以一个位平面为内存中的一个块，此块通过 VGA 图形控制器芯片从内存中段 0xA000 开始映射到屏幕上。适于映射到用户物理屏幕内存中的位平面数目越大，用户的 VGA 视频适配器可支持的颜色数目越大。这就是为什么 VGA 视频卡可以处理各种数量的内存。VGA 卡内存越大，用户所拥有的可用于映射到内存中的位平面数越大，用户所拥有的颜色也就越多。这不是很简单。

现在，VGA 视频位平面结构体系是将一图形显示映射到内存中的最好方式。这就是我们正在处理的方式。

1.2 什么是位平面动画

现在我们已经理解什么是位平面，让我们看看位平面动画。动画，正是本书的目的，意味着屏幕上的图形对象的移动。本书中使用的位平面动画技术允许用户以各种速度且平滑地移动三种类型的固定尺寸图形对象。这些图形对象类型为：

图形对象名称	说 明
小精灵(Sprite)	通过子图形编辑器定义的 47×46 图形对象
导弹(Missile)	小圆球对象(固定的对象与尺寸)
子弹(Bullet)	更小的圆球对象(固定的对象与尺寸)

所以，作为本书的目的，位平面动画意味着小精灵、导弹和子弹能在屏幕上高速、平滑地移动。

1.3　位平面动画是如何完成的

这里是对位平面动画如何完成的一个简略介绍：

1. 选择(或映射)使用 VGA 图形控制器芯片的 VGA 位平面。

2. 直接将图形对象(小精灵、导弹、子弹)写到物理屏幕内存中的合适地址中。

然而，本书中使用的实际位平面动画制作过程远比这一过程复杂。尽管在此是关于位平面动画制作技术，我将尽可能描述得详细些，但是，你将知道应用编程者使用以文本描述的小精灵动画制作视频函数集时，并不需要知道位平面动画的错综复杂的关系。

作为本书的目的，图形对象在其上移动的屏幕背景称为游戏域。写图形对象到屏幕上的方法可以描述为在游戏域上写图形对象。因此，如果你希望在游戏域上移动图形对象，只需简单地在此游戏域上写此图形对象即可。

问题是在此游戏域中写图形对象的过程，事实上是使图形的重复图像可见，而这又不是希望的效果。在本书中用于解决这一问题的技术是将此图形对象写到屏幕上，让其显示足够长的时间，以便人眼能获得此信息，将其从屏幕上移去，然后在相邻的屏幕处定位，沿希望的方向重写此图形对象到屏幕上。动画制作以下列方法创建：

1. 将图形对象写到游戏域位平面上。

2. 让此对象以某一特定时间显示。

3. 将此图形对象从屏幕上移去。

4. 在相邻的屏幕定位处，沿希望的方向重写此图形对象。

OK，你现在可能会问："如何删除图形对象而不在游戏域留下斑点？"这是一个好问题。

有几种方法可以处理这一图形对象删除问题。我选择了一种简单且有条理的方法，尽管此方法有某种取舍的意味。通过写一位平面动画制作初始化程序，我决定了这一问题。这一初始化程序存储 4 个位平面的所有屏幕内存，而这些内存保存了游戏域中的所有信息。当你删除一图形对象，你实际上是将原游戏域数据拷贝到相近的位平面上。原理就是这样，是不是很简单？

下面也是一种取舍。如果在同一位平面移动的两图形对象互相覆盖，将出现什么现象？将有某些对象产生晃动现象，对象移动的越快，人眼注意的时间越短。现在，通过将游戏域屏幕数据存储到内存中，可以消去这一特殊条件，然后绘制小精灵，接着通过调用先前存储的游戏对象将小精灵删去，但是，我并没有这样做，下面就解释其原因：

当我编写演示程序时，要求有半打以上的对象在屏幕上高速平滑移动。图形对象的重复存储减慢了这一工作的速度。所以，我选择了在特定的和可预见的条件下能减小对象晃动的移动速度。

1.4　设计小精灵

你可以以两种方法设计小精灵。其一你可以使用磁盘上提供的子图形编辑器，它的源码列在本书第三部分。子图形编辑器允许用户通过使用鼠标打开和关闭单个子图形像素。通过此编辑器，你可以设计出 1 到 4 个带不同颜色的子图形。当你将子图形对象存到磁盘上

时，它是以 C 源码的方式存储的。

描述此小精灵图形的 C 源码可以正常被编译，且可用在用户的程序中。这是很简单的方式，更困难的方式涉及到用 C 或汇编语言手工编码小精灵对象。如果你使用. RADIX 运算符，且将其设置为 2，此汇编方法是很容易的，然而，我可以告诉你一旦小精灵编辑器的原型完成，且在我的环境中运行，我就再也不用手工编码一小精灵对象。

1.5　小　　结

位平面是内存中的一个域，该域可映射到从段地址 0xA000 开始的物理内存中。你所拥有的位平面越多，显示所拥有的颜色选项也就越多。本书中的程序专门设计在 640×480 16 色 VGA 视频模式环境下工作。

位平面动画制作指的是在某个视频平面上图形对象（小精灵、导弹和子弹）的平滑移动。本书中给出的动画函数对速度的选择超过其他方面的考虑。这一速度的选择意味着在同一平面上运动的图形对象相互覆盖时，涉及到潜在的小型对象抖动被取舍。

本书中提供的小精灵编辑器用于加工你的小精灵开发。小精灵对象以 C 源码文件存储在硬盘上，且可以被编译并放置在一个库中，便于自己的使用。

第二章开始给出归档的位平面样本程序。此章中你会学到如何使你的计算机沿水平、垂直和对角线方向移动一个小精灵。

第二章　在游戏域中移动小精灵

本章提供一些基本信息，这些信息将对用户使用本书中给出的小精灵管理函数有所帮助。小精灵管理创建过程与通用小精灵管理函数一起给出。同时给出两个小精灵动画示范程序。

第一个示范程序(PROG2-1.C，程序 2-1)向用户说明如何移动一个小精灵，此程序可用 Borland C/C++ 3.1 版编译器编译。紧接其后的是 Borland MAKE 文件(程序 2-2)。第二个示范程序(PROG2-2.C)也说明小精灵的动画制作，此程序用 Microsoft C/C++编译器编译。Microsoft MAKE 文件(程序 2-4)紧接其后。

下面给出 Borland 和 Microsoft 示范程序，从此程序中，你将找到一正式的基本管理函数。用户将学会如何将小精灵对象集成到你自己的源文件中，如何扩展小精灵对象以便适应所有水平像素定位，如何初始化屏幕的游戏域以及破坏此游戏域。

2.1　Borland GI 无缝界面的示范

程序 2-1 给出了列在 PROG2-1.C 中的源码。这一程序被设计用 Borland C/C++编译器来编译。通过测试此文件，你可以发现把小精灵管理函数集成到 Borland 图形环境中是多么方便。

程序 2-1　列在 PROG2-1.C 中的源码

```
//////////////////////////////////////
//
// prog2－1.c
//
// Demonstration of simple movement of
// one sprite demonstrating a seamless
// interface to Borland's Graphical
// Interface (BGI)
//

//////////////////////////////////////
//
// defines
//

#include <graphics.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdio.h>
#include <string.h>
```

```
#include <conio.h>
#include "tproto.h"
#include "sprite.h"

////////////////////////////////////////
//
// include sprite created by the
// sprite editor

#include "BOX.C"

void main(void);
void __delay(int num);

SPRITE_IMAGE sprite1;

void main()
{
int row, col, key, cnt;
int param;
int gdriver= DETECT, gmode, errorcode;
int midx, midy, mode;
int max_x, max_y;

    // BGI graphics initialization routine

    initgraph(&gdriver, &gmode, "");

    // check to see if there is an error

    errorcode= graphresult();

    // if initialization error occurs then report
    // error and abort

    if(errorcode != gr0k) {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any to return to DOS\n");
      gtKey();
      exit(1);
      }

    // expand image in buffer

    expand_sprite_image(BOX1);

    // initialize the sprite

    init_sprite(&sprite1, BOX1, 1);

    setbkcolor(LIGHTGRAY);
```

```
    // set text style

    settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);

    // print message to screen in graphics text

    setcolor(BLUE);

    outtextxy(140, 20, "ENTER Runs Program -- F10 Quits to DOS");

    // initialize graphics playfied after playfield
    // is drawn

    init_playfield();

    // initialize row and column location of sprite

    row= 50;
    col= 50;

    // display sprite image on screen

      move_sprite_image(col, row, &sprite1);

    // wait for key press

    key= gtKey();

    if(key == F10) {
      goto program_abort;
      }

    ////////////////////////////////////
    // goto program execution label

restart_program:

    ////////////////////////////////////
    // move sprite horizontally right

    for(;;) {
      if(col > 560) {
        break;
        }

      // move the sprite image

          move_sprite_image(col++, row, &sprite1);

      // slow down to human time frame
```

```
    delay(12);
  }

////////////////////////////////////
// move sprite down

for(;;) {
  if(row >= 400) {
    break;
    }

  // move the sprite image

    move_sprite_image(col, row++, &sprite1);

  // slow down to human time frame

    delay(12);
  }

////////////////////////////////////
// move sprite horizontallty left

for(;;) {
  if(col <= 50) {
    break;
    }

  // move the sprite image

    move_sprite_image(col--, row, &sprite1);

  // slow down to human time frame

    delay(12);
  }

////////////////////////////////////
// move sprite up

for(;;) {
  if(row <= 50) {
    break;
    }

  // move the sprite image

      move_sprite_image(col, row--, &sprite1);

  // slow down to human time frame
```

```
    _delay(12);
    }

  ////////////////////////////////////
  // move sprite diagonally down
  // right

  for(;;) {
    if(row > 400) {
      break;
      }

    // move the sprite image

        move_sprite_image(col++, row++, &sprite1);

    // slow down to human time frame

    _delay(12);
    }

  ////////////////////////////////////
  // move sprite diagonally up
  // left

  for(;;) {
    if(row < 50) {
      break;
      }

    // move the sprite image

      move_sprite_image(col--, row--, &sprite1);

    // slow down to human time frame

    _delay(12);
    }

  key= gtKey();

  if(key != F10) {
    goto restart_program;
    }

  // program has been aborted before it is run

program_abort:

  // return from graphics mode
```