Appendix F

Graphics: Simulating movement and keyboard event handling

F.1 Overview

This appendix explains how to create the fast movement of mutiple graphical primitives. It also explains how you can detect keyboard events using allegro. This is much more convenient that using getch which pauses the program. It is only relevant if you are using the graphics library `graphics_lib.h`.

F.2 Movement

We have already seen how the appearance of movement can be made. The principle is this. We draw a graphical element on an area of the display and then subsequently we paint over it with the same colour as the display background. We can paint over using any the “filled” functions: `filled_circle`, `filled_ellipse`, `filled_rectangle`. Alternatively we can used the `cleardevice` function, which clears the entire display area by paiting over it with the background colour.

To give the appearance of multiple graphical elements moving we simply need to draw multiple elements and re-draw them in a new position and paint over the previous drawn elements with the screen background colour. We will discuss in detail a small program that gives the appearance of two balls endlessly bouncing off walls. Often when you have simulated movement, you need some user input that can decide to stop the movement or carry out another action. The example program will allow this by having a mouse event handler that checks if the user has pressed the right mouse button. If so, it will halt the movement of the balls, resuming their movement after the right mouse button is released. It will also shut down the graphics window when the left mouse button is pressed and end the while-loop in preparation for terminating the program.

Finally, the program introduces keyboard event handling in allegro. This allows detection of keyboard events while the ball is moving, without causing any delay in the movement (unlike using getch).

F.3 Examining the program “events.c”

Assuming you have the “clab” project open in Code:: Blocks change the target to “events” and open the source code for the mouse program “events.c”. We will examine this program now. The program begins with some include statements which allow the use of the graphics wrapper functions defined in `graphics_lib.h` and a few other libraries in the standard C library. Then it defines some useful global constants that are used by the program.

```
#include <graphics_lib.h>
#include <stdio.h>
#include <conio.h>
```

```
#include <math.h>

#define XWINDOW 640
#define YWINDOW 480
#define TICK    0.5
#define RADIUS  10
#define XMAX    XWINDOW - RADIUS
#define XMIN    RADIUS
#define YMAX    YWINDOW - RADIUS
#define YMIN    3*RADIUS
#define MAXCOUNT 10000

/* make a data type hold position and velocity */
typedef struct
{
   double    pos;
   double    vel;
}  posvel;

posvel hit_boundary(double pos, double vel, int isx);
```

It defines a data structure called `posvel` which allow variables to be used that hold two double precision numbers accessed via the structure member variables `pos` and `vel`. The data structure is very useful in a function called `hit_boundary`. The function `hit_boundary` checks when the two balls encounter a boundary. If they do, then they are made to appear to bounce of the walls. This requires a change to their position and velocity. The `hit_boundary` function is shown below:

```
/* handle what happens to balls at boundaries of the graphics window */
posvel hit_boundary(double pos, double vel, int isx)
{
   posvel z;

   z.pos = pos;
   z.vel = vel;

   if (isx)
   {
      if (pos < XMIN)
      {
         z.pos = XMIN;
         z.vel = -vel;
      }
      else if (pos > XMAX)
      {
         z.pos = XMAX;
         z.vel = -vel;
      }
   }
   else
   {
      if (pos < YMIN)
      {
         z.pos = YMIN;
         z.vel = -vel;
      }
      if (pos > YMAX)
      {
         z.pos = YMAX;
         z.vel = -vel;
      }
   }

   return z;
}
```

The parameter `isx` which takes the values 0 or 1 is used to choose whether the boundary apply on the vertical or horizontal boundaries. If the balls do not hit boundaries then the function just returns the original values of position and velocity supplied by the calling function (in this case, the `main` function).

The `main` function is given below:

```
int main(void)
{
    int     count = 0;
    int     G_pressed = 0, R_pressed = 1;
    char    vstring[10], wstring[10];
    double  x1_old, y1_old;
    double  x2_old, y2_old;
    double  x1_new, y1_new;
    double  x2_new, y2_new;
    double  vx, vy, wx, wy;
    double  v, w;
    posvel  pv;

    x1_old = 100.0; y1_old = 100.0;
    vx = 10.0; vy = 5.0;

    x2_old = 250.0; y2_old = 250.0;
    wx = -8.0; wy = 4.0;

    v = sqrt(pow(wx,2)+pow(wy,2));
    w = sqrt(pow(vx,2)+pow(vy,2));

    /* open the graphics window */
    initwindow(XWINDOW, YWINDOW);

    /* allow mouse operations */
  initmouse();

    /* allow keyboard operations */
    initkeyboard();

    /* create an event queue */
    create_event_queue();

    /* register display, mouse and keyboard as event sources */
    reg_display_events(); reg_mouse_events(); reg_keyboard_events();

    /* initialize the font */
    initfont();

    outtextxy(4,5,"To quit press left mouse button or close graphics window");
    outtextxy(4,15,"To pause press right mouse button");
    outtextxy(4,25,"To change ball speed use arrow keys (up/down = red, left/right = 
    outtextxy(4,35,"To make the red ball green press G or g ");
```

```
do
{
       if (check_if_event())
       {
           /* wait for event  */
           wait_for_event();

           if (event_close_display())
               break;
           else if (event_mouse_button_down())
           {
               if (event_mouse_left_button_down())
                   break;
               else if (event_mouse_right_button_down())
                   wait_for_event();
           }
           else if (event_key_down())
           { /* change speed of first ball */
               if(event_key_up_arrow())
               {
                   vx = 1.25*vx;
                   vy = 1.25*vy;
               }
               else if(event_key_down_arrow())
               {
                   vx = 0.75*vx;
                   vy = 0.75*vy;
               }
               else if(event_key_left_arrow())
               {
                   wx = 0.75*wx;
                   wy = 0.75*wy;
               }
               else if(event_key_right_arrow())
               {
                   wx = 1.25*wx;
                   wy = 1.25*wy;
               }
               if (event_key('G'))
               {
                   G_pressed = 1;
                   R_pressed = 0;
               }
               else if (event_key('R'))
               {
                   R_pressed = 1;
                   G_pressed = 0;
               }
           }
       }
       v = sqrt(pow(wx,2)+pow(wy,2));
       w = sqrt(pow(vx,2)+pow(vy,2));
     /* calculate new ball positions */
       x1_new = x1_old + vx*TICK;
       y1_new = y1_old + vy*TICK;
       x2_new = x2_old + wx*TICK;
       y2_new = y2_old + wy*TICK;

       /* handle what to do if balls hit boundaries */
```

```
      pv = hit_boundary(x1_new, vx, 1);
      x1_new = pv.pos;
      vx = pv.vel;

      pv = hit_boundary(x2_new, wx, 1);
      x2_new = pv.pos;
      wx = pv.vel;

      pv = hit_boundary(y1_new, vy, 0);
      y1_new = pv.pos;
      vy = pv.vel;

      pv = hit_boundary(y2_new, wy, 0);
      y2_new = pv.pos;
      wy = pv.vel;

      /* draw balls on screen buffer in new positions */
      filled_circle(x1_new, y1_new, RADIUS , BLUE);
      if (G_pressed)
          filled_circle(x2_new, y2_new, RADIUS , GREEN);
      else if (R_pressed)
          filled_circle(x2_new, y2_new, RADIUS , RED);

      /* make the balls visible on the screen display
         and remove the balls in the previous positions */
    sprintf(vstring, "%4.2lf", v);
    sprintf(wstring, "%4.2lf", w);
      setcolor(RED);
      outtextxy(VEL_TEXT_X,VEL_TEXT_Y,"v =  ");
      /* clear area for numeric output */
      filled_rectangle(VEL_TEXT_X+40, VEL_TEXT_Y-10, VEL_TEXT_X+100, VEL_TEXT_Y+20,
      outtextxy(VEL_TEXT_X+60,VEL_TEXT_Y,vstring);
      setcolor(BLUE);
      outtextxy(VEL_TEXT_X+120,VEL_TEXT_Y,"w =  ");
      /* clear area for numeric output */
      filled_rectangle(VEL_TEXT_X+160, VEL_TEXT_Y-10, VEL_TEXT_X+220, VEL_TEXT_Y+20
      outtextxy(VEL_TEXT_X+180,VEL_TEXT_Y,wstring);

      update_display();

      /* remove the balls in the previous positions on
         the screen buffer */
      filled_circle(x1_old, y1_old, RADIUS , BLACK);
      filled_circle(x2_old, y2_old, RADIUS , BLACK);

      /* update the old positions */
      x1_old = x1_new;
      y1_old = y1_new;
      x2_old = x2_new;
      y2_old = y2_new;

      count++;
      pausefor(8); /* wait 8 miliseconds */


}
  while (count < MAXCOUNT);

  /* close the mouse */
```

```
    closemouse();

    /* remove the display */
    closegraph();

    return 0;
}
```

If you are familiar with appendix E.1 which discussed how to register display and mouse events, you will be familar with many of the functions within the `main` function. However several new functions have been introduced. The first is a function that allows the execution of a program to pause. The graphics library "graphics_lib.c" has its own pause function called `pausefor`. It is defined below:

Function Reference: `pausefor` — Waits for a specified amount of time

`pausefor(`*`duration`*`);`

The `pausefor` function causes the computer to wait before continuing. The amount of time the computer waits for is determined by the *`duration`* argument, which is a integer number, and specifies the amount of time the computer should wait for in one thousandths of a second (milliseconds).

For example:

```
    pausefor(500);
```

will cause the computer to wait for half a second (500 milliseconds) before continuing.

`pausefor` is defined in `graphics_lib.h`.

Another new function `check_if_event` is extremely useful in a situation where the computer is executing a loop but you want it to be able to be interupted without causing any appreciable slowing down of the loop. You see the idea of the bouncing balls is that you want them to continue doing this until the user carries out a specific action. In this case the action is either closing the display or clicking a mouse button, or detecting what keys have ben pressed on the keyboard.

Function Reference: `check_if_event` — Detect whether any new event has been added to the event queue

`check_if_event();`

The int `check_if_event` returns 1 if a registered event source has encountered an event, otherwise it returns zero.

`check_if_event` is defined in `graphics_lib.h`.

Although it was not used in this program another useful function is one that can add random variation. For instance we could have made the balls bounce off the walls at random angles. The function is called `rand_number` and is defined in the graphics library `graphics_lib.h` and the equivalent function `random_number` is defined in `amio_lib.h`. The definitions of these function were given in appendix C and lab 4.

Other functions introduced in the program are:

Function Reference: `reg_keyboard_events` — Allow keyboard events to be detected

```
reg_keyboard_events();
```

The void `reg_keyboard_events` function allows keyboard events to be detected.

`reg_keyboard_events` is defined in `graphics_lib.h`.

Function Reference: `event_key_down` — Detect whether any keyboard key has been pressed

```
event_key_down();
```

The int `event_key_down` returns 1 if any key on the keyboard has been pressed, otherwise it returns zero.

`event_key_down` is defined in `graphics_lib.h`.

The following functions are more specific and test whether a particular key on the keyboard has been pressed.

Function Reference: `event_key_up_arrow` — Detect whether the up arrow key has been pressed

```
event_key_up_arrow();
```

The int `event_key_up_arrow` returns 1 if the up arrow key has been pressed, otherwise it returns zero.

`event_key_up_arrow` is defined in `graphics_lib.h`.

Function Reference: `event_key_down_arrow` — Detect whether the down arrow key has been pressed

```
event_key_down_arrow();
```

The int `event_key_down_arrow` returns 1 if the down arrow key has been pressed, otherwise it returns zero.

`event_key_down_arrow` is defined in `graphics_lib.h`.

Function Reference: `event_key_left_arrow` — Detect whether the left arrow key has been pressed

```
event_key_left_arrow();
```

The int `event_key_left_arrow` returns 1 if the left arrow key has been pressed, otherwise it returns zero.

`event_key_left_arrow` is defined in `graphics_lib.h`.

Function Reference: `event_key_right_arrow` — Detect whether the right arrow key has been pressed

```
event_key_right_arrow();
```

The int `event_key_right_arrow` returns 1 if the right arrow key has been pressed, otherwise it returns zero.

`event_key_right_arrow` is defined in `graphics_lib.h`.

Function Reference: `event_key` — Detect whether a character key has been pressed

```
event_key(char letter);
```

The int `event_key` returns 1 if the letter supplied as an argument (case insenstive) has been pressed, otherwise it returns zero.

For example

```
    event_key('G');
```

detects whether either the 'g' or 'G' keys have been pressed. In the porgram above, this is used to change the colour of one of the bouncing balls.

`event_key` is defined in `graphics_lib.h`.

F.4 Running and altering the bouncing balls program "events.c"

Build and execute the program "events.c". Notice how the upper horizontal boundary is invisible and prevents the balls from overwriting the text which explains how to stop the program. Try pressing the right mouse button. It pauses the movement of the balls and as soon as it is released the balls start moving again. The smoothness of the ball motion (i.e. the number of times a ball is drawn is determined by the `TICK` constant. While the apparent speed at which the balls move is determined by a combination of TICK and the pause time (see `pausefor`).

F.5 Summary

After reading through this appendix and running the events program, you should have a good idea how to simulate the movement of multiple graphical entities in your programs. You will also have learned how to detect keyboard events using the allegro graphics library. This should allow you to build interesting games and graphical interfaces.