

Computer Labs: Lab 5 & Sprites

2º MIEIC

Pedro F. Souto (pfs@fe.up.pt)

November 17, 2014

Pixmap and XPM

pixmap is short term for “pixel map”, the representation of a graphic image as an array of pixel color values

- ▶ I.e. it is a map of screen coordinates to color values
- ▶ **bitmap** is a pixmap that uses a single bit to denote the color of each pixel

XPM X Pixmap is an image format that allows to represent a pixmap in a textual form, by representing each color value by a different character (actually, may be more than one per color)

- ▶ An XPM for a given pixmap can be stored either in a text file, or in a data structure of a C program

Lab 5: test_xpm()

```
test_xpm(unsigned short xi, unsigned short yi,  
         char *xpm[])
```

What Display the XPM provided in `xpm` at the screen coordinates `(xi, yi)`

- ▶ Use VBE mode `0x105`

Issue How to convert the XPM to a pixmap?

Answer Use the `read_xpm()` function

Generating a Pixmap from its XPM: `read_xpm()`

```
int width, height;
char *map;

// get the pix map from the XPM
map = read_xpm(pic1, &width, &height);

// copy it to graphics memory
```

```
char *read_xpm(char* pic1, int *w, int *h)
```

reads an XPM description of a pixmap `pic1`, and returns the pixmap as a two-dimensional char array of `*h` lines, each of which with `*w` pixels. It assumes that the XPM uses:

- ▶ One char per color – this is enough if we have few colors
- ▶ One byte per color – this is OK for mode `0x105`

The “Class” Sprite: `sprite.h`

Sprite “Two-dimensional image that is integrated into a larger scene” (Wikipedia)

- ▶ Allows the integration of independent pixmaps into a scene
- ▶ Allows image animation without altering the background – thus a sprite can be considered an overlay image

```
typedef struct {  
    int x, y; // current position  
    int width, height; // dimensions  
    int xspeed, yspeed; // current speed  
    char *map; // the pixmap  
} Sprite;
```

The pixmap uses **black** for the background, which is assumed to be transparent

The “Class” Sprite: `sprite.c`

```
/** Creates a new sprite with pixmap "pic", random speeds
 * (not zero) and position (within the screen limits), and
 * draws it in memory whose address is "base";
 * Returns NULL on invalid pixmap.
 */
Sprite *create_sprite(char *pic[], char *bas) {

    //allocate space for the "object"
    Sprite *sp = (Sprite *) malloc ( sizeof(Sprite));
    if( sp == NULL )
        return NULL;

    // read the sprite pixmap
    sp->map = read_xpm(pic, &(sp->width), &(sp->height));
    if( sp->map == NULL ) {
        free(sp);
        return NULL;
    }
    ...
    return sp;
}
```

The “Class” Sprite: `sprite.c`

```
void destroy_sprite(Sprite *sp, char *base) {
    if( sp == NULL )
        return;
    free(sp->map);
    free(sp);
    sp = NULL;      // hopeless: pointer is passed by value
}

int animate_sprite(Sprite *sp, char *base) {
    ...
}

/* Some useful non-visible functions */

static int check_collision(Sprite *sp, char *base) {
    ...
}
static int draw_sprite(Sprite *sp, char *base) {
    ...
}
```


Lab 5: test_move()

```
int test_move(unsigned short xi, unsigned short yi,  
             char *xpm[], unsigned short hor,  
             short delta, unsigned short time)
```

What? Move a sprite on the screen

`(xi, yi)` initial coordinates

`xpm` XPM for the sprite

`hor` if non-zero horizontal movement, otherwise vertical

`delta` movement length: negative if in decreasing coordinates
direction

`time` time to execute the movement in seconds

How? Should use the `sprite` "class"

- ▶ But you can change it slightly (I did).
- ▶ Need not implement all functions.

Sprite Animation

- ▶ Animation of a sprite can be achieved by presenting a sequence of pixmaps
 - ▶ Each pixmap (but the first) in this sequence differs slightly from the previous pixmap



- ▶ To create an animated sprite we need to specify several pixmaps
 - ▶ This can be done in different ways
- ▶ We'll use a C function with a variable number of arguments:

```
AnimSprite *create_animSprite(char *base,  
                               char *pic1[], ...);
```

`printf()` is the most common C function of this type

(Functions with a Variable Number of Arguments)

- ▶ Must have at least one argument
- ▶ Need to know how many arguments
 - ▶ Either the number is specified in one of the required arguments
 - ▶ Or the last argument has a special value, e.g. NULL
- ▶ Uses a list of variable arguments of type `va_list`
- ▶ Relies on a set of macros defined in `<stdarg.h>`, which implement a kind of iterator for accessing that list:

`va_start` to initialize the list

`va_arg` to access the next argument (list element)

`va_end` to finalize the access

```
#include <stdarg.h> // va_* macros are defined here
int foo(int required, ...) {
    va_list var_args;
    va_start(var_args, required);
    int i = va_arg(var_args, int);
    float f = va_arg(var_args, float);
    char *s = va_arg(var_args, char *);
    va_end(var_args);
}
```

The “Class” Animated Sprite: AnimSprite.h

```
#include <stdarg.h> // va_* macros are defined here
#include "sprite.h"
typedef struct {
    Sprite *sp;          // standard sprite
    int aspeed;         // no. frames per pixmap
    int cur_aspeed;     // no. frames left to next change
    int num_fig;        // number of pixmaps
    int cur_fig;        // current pixmap
    char **map;         // array of pointers to pixmaps
} AnimSprite;
```

```
AnimSprite(char *base, char *pic1[], ...);
int animate_animSprite(AnimSprite *sp, char *base);
void destroy_animSprite(AnimSprite *sp, char *base);
```

Animation speed is measured as number of “frames” per pixmap

The “Class” Animated Sprite: AnimSprite.c (1/2)

```
AnimSprite *create_animSprite(char *base, char *pic1[], ...) {
    AnimSprite *asp = malloc(sizeof(AnimSprite));
    // create a standard sprite with first pixmap
    asp->sp = create_sprite(pic1, base);
    // find out the number of variable arguments
    va_list var_args; // variable arguments
    int args;
    // find out the length of the va_args list
    va_start(va_args, pic1); // initialize va_args list
    // iterate over that list
    for(args = 0; va_arg(var_args, char**) != NULL; args++);
    va_end(va_args); // done with va_args list, for now
    // allocate array of pointers to pixmaps
    asp->map = malloc((args+1) * sizeof(char *));
    // initialize the first pixmap
    asp->map[0] = asp->sp->map;
    // continues in next transparency
```

The “Class” Animated Sprite: AnimSprite.c (2/2)

```
// initialize the remainder with the variable arguments
// iterate over the var_args list again
va_start(var_args, pic1);
for( i = 1; i <args+1; i++ ) {
    char **tmp = va_arg(var_args, char **);
    asp->map[i] = read_xpm(tmp, &w, &h);
    if( asp->map[i] == NULL
        || w != asp->sp->width || h != asp->sp->height) {
        // failure: release allocated memory
        for(j = 1; j<i;j ++){
            free(asp->map[j]);
        }
        free(asp->map);
        destroy_sprite(asp->sp);
        free(asp);
        va_end(var_args);
        return NULL;
    }
}
va_end(var_args);
...
}
```

Thanks to:

I.e. shamelessly translated material by:

- ▶ João Cardoso (jcard@fe.up.pt)

Further Reading

- ▶ João Cardoso, [Notas sobre *Sprites*](#)