Organización del Computador II

Primer Cuatrimestre de 2008

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo práctico I

Rescatando a la Princesa Peach

Abstract

Modificación de BMPs píxel a píxel, comparaciones en asembler y otras yerbas para la confección de un escenario de Mario Bross.

Palabras clave

Mario, Princesa, Asembler, Caño, Moneda, asm, BMPs, byte, RGB.

Grupo 15

Integrante	LU	Correo electrónico
Cristian Alejandro Archilla	433/03	gnigel@gmail.com
Matías López y Rosenfeld	437/03	matiaslopez@gmail.com
Tomás Scally	886/03	horizontedesucesos@gmail.com

Índice

1. Introducción

Para la realización de este trabajo práctico nos fue necesario, entre otras cuestiones logísticas, aprender a lidiar con las instrucciones de *assembler*. Esta tarea no nos resultó sencilla en un primer momento, pero gracias a que mientras hacíamos tp, también estudiabamos para el parcial, fuimos aprendiendo rapidamente.

Uno de los puntos más importantes que cabe destacar en la realización de este trabajo práctico es la gran comodidad que nos permitió para realizarlo la posibilidad de contar con un *svn* para no tener que perder tiempo intercambiandonos la última versión y buscando versiones viejas. Esto nos resultó muy cómodo ni bien empezamos a programar, cuando de golpe tras cambiar 3 o 4 lineas el programa daba Segmentation Fault ("*Segfaulteaba*").

Como metodología de trabajo no utilizamos ninguna en particular más que ir haciendo las funciones en el orden que figuraban en el enunciado. Esta "decisión" se ve reflejada en la calidad del código. Las últimas funciones aprovechan mucho más las operaciones del lenguaje. Desgraciadamente no contamos con tiempo como para rehacer el trabajo práctico, ya que sin duda si encararamos nuevamente las funciones nos saldrían distintas. Esto refleja nuestro aprendizaje a lo largo del desarrollo del tp.

Salvo las funciones **checkcolision** y **salta**, las demás (**apagar**, **generarFondo**, **blit**, **recortar**) son funciones que modifican directamente la estructura de un píxel. Las 2 primeras son puramente numéricas que realizan algunos cálculos y comparaciones.

Brevemente podemos describir el funcionamiento de estas 6 funciones así (en orden de realización):

- **generarFondo**: Pinta el cielo y copia el piso.
- **recortar**: Recorta una parte de un sprite.
- **blit**: Saca el color de off para mimetizar la imagen con el fondo.
- **checkcolision**: Comprueba si hay colisión entre 2 objetos, en este caso, **Mario** y el caño o la caja de las monedas.
- **salta**: Maneja el comportamiento de **Mario** cuando salta.
- **apagar**: Va cambiando el color de las monedas.

2. Desarrollo

En esta sección describiremos una por una las funciones implementadas a lo largo de todo el trabajo, explicaremos los *trucos* usados para ciertos cálculos, como por ejemplo el ancho de la imagen en bytes.

1. generarFondo:

   Esta función está dividida en dos etapas, una para pintar el cielo y otra para generar el piso. Para la primer etapa utilizamos dos ciclos para recorrer la imagen, uno para las columnas y otro para las filas en cada píxel grabamos con el color del cielo.

   En la segunda etapa de la función nos ocupamos de generar el piso. Para esto vamos copiando píxel a píxel el sprite correspondiente al piso varias veces una al lado de la otra hasta completar todo el ancho de la pantalla.

   En esta función no calculamos la basura sino que avanzamos los píxels que nos faltaban para completar la fila de la imagen *a mano*.

2. RECORTAR:

   En esta función nuestro código ya evolucionó un poco y comenzamos a utilizar los registros con 4 bytes. Para calcular la basura hacemos la cuenta por fila y ya dejamos de utilizar el famoso método de "prueba y error" al que recurrimos en la función **generarFondo**. La cuenta es

   $$\frac{TamañoImagen \times 3}{4}$$

   donde $TamañoImagen$ está en píxels. Si el resto es distinto de cero, entonces

   $$Basura = 4 - Resto$$

   Si el resto es cero no hay basura. Luego lo que hacemos es recorrer en la imágen original el sprite que me pide la función y vamos escribiendo por fila en el destino y cuando completamos la fila le sumamos la $Basura$ necesaria para completar los 4 bytes.

3. BLIT:

   Esta función reemplaza el color *fucsia* (**0xFF00FF**) por el color de fondo. Acá también trabajamos con registros de cuatro bytes, calculamos la basura como en el caso anterior y vamos reemplazando cuando tenemos un píxel color *fucsia* por el color *azul* (**0x0096FF**) del fondo.

   Al trabajar con cuatro bytes tenemos en cuenta que estamos agarrando uno que no nos corresponde y volvemos a escribir lo que estaba originalmente en esa posición.

   Para el píxel inicial, tenemos que mover el puntero un byte hacia atrás de modo de cargar en los bytes menos significativos el primer píxel. ¿Por qué hacemos esto? Porque como el procesador trabaja en *little-endian*, necesitamos este truquito para alinear los primeros tres bytes de la imagen (el 1er píxel) con los bytes más significativos del registro en el que lo vamos modificando.

4. CHECKCOLISION:

   Esta es la función que nos resultó más simple en todo el TP. La resolvimos con cuatro comparaciones, dos juegos de comparaciones iguales para cada eje. Si la coordenada más chica del 1er objeto a comparar es menor que la más grande del 2do, ya no hay colisión y viceversa. Esto para cada eje.

   Es decir, si tenemos los objetos **A** y **B** con sus respectivos puntos x, xw, y y yh, con $x < xw$ e $y < yh$. Hay colisión sólo si todas las comparaciones siguientes dan falso:

   - $Ax > Bxw$
   - $Axw < Bx$
   - $Ay > Byh$
   - $Ayh < By$

5. SALTA:

   Esta función implementa el algoritmo provisto por la cátedra en el enunciado. Como detalle, no fue necesario utilizar variables locales ya que en los registros pudimos salvaguardar, para almacenar y utilizar todos los valores a tener en cuenta para realizar esta función.

6. APAGAR:

   Esta función también consiste en modificar una imágen, usando la misma técnica para calcular la basura que en las funciones anteriores. Cuando encontramos un píxel que corresponde a la moneda, tenemos que reemplazarlo por datos que vienen como parámetro. Una de las cosas cuyo estado no nos enorgullece en este TP, es que en esta función traemos de la memoria el puntero al contador cada vez que reemplazamos un píxel. Esto se podría mejorar reutilizando registros.

3. Conclusiones

A lo largo del trabajo pudimos entender en profundidad el concepto de little-endian. Al tener que trabajar con tres bytes utilizando registros de cuatro nos ha dado varios dolores de cabeza a la hora de leer y escribir píxeles. Creemos que este fue uno de los obstáculos mas importantes del trabajo práctico.

Para nuestra sorpresa, pudimos ver que pasar pseudocódigo al *assembler* se puede utilizar un orden similar al de lenguajes de mas alto nivel, pero sin perder de vista las restricciones que ofrece assembler (cantidad de registros, operaciones, etc).

Inicialmente generamos todas las funciones vacias y a medida en que fuimos implementandolas se noto una baja de performance considerable. Esto se veia claramente en la velocidad en la que camina **Mario** a lo largo de la pantalla y en especial en las máquinas del laboratorio 5 del DC (*Labo Linux*) siendo estas “máquinas de ultima generación”.

También pudimos ver que no podemos fiarnos de los resultado gráficos hasta que no implementamos integramente todo el tp. Por ejemplo, al terminar **checkColision** cuando **Mario** colisionaba con el pipe se subia a este y luego seguia caminando en el aire. Esto nos hizo pensar mucho tiempo que el problema estaba en la implementación de **checkColision** hasta que decidimos seguir e implementar **salta** después de lo cual nos alegramos enormemente al ver que **Mario** hacia lo que tenia que hacer. Así que con la satisfacción del deber cumplido nos tomamos una birra para festejar.

Teniendo en cuenta los problemas que han tenido algunos de nuestros compañeros en windows podemos concluir que trabajar bajo la plataforma de linux nos ha ahorrado muchos de estos problemas y ha sido la mejor opción de trabajo.

4. Manual del usuario

4.1. Detalles del contenido del cd

En el **CD** usted podrá ver 6 archivos .asm cada uno se llama como la función cuyo codigo guarda en su interior, un archivo Makefile para poder compilar el tp, un archivo .c y un directorio imagenes con su contenido, estos últimos 2 provistos por la cátedra.

El listado de archivos sería:

```
apagar.asm
blit.asm
checkcolision.asm
generarFondo.asm
main.c
Makefile
recortar.asm
salta.asm

imagenes:
cloud.bmp
coin.bmp
marioJump.bmp
marioJumpL.bmp
marioSpriteW.bmp
marioSpriteWL.bmp
marioStand.bmp
mon.bmp
pipe.bmp
piso.bmp
PrincessToadstool.bmp
questB.bmp
quest.bmp
victoria.bmp
```

4.2. Uso del contenido del cd

Dentro del material entregado en el medio extraible digital que adjuntamos a este informe, cuenta usted con un **Makefile**. ¿Qué quiere decir eso? Quiere decir que si ud., en la misma plataforma en la que nosotros trabajamos, escribe la palabra "make" (en el directorio donde está el archivo **Makefile**) y presiona enter (↵), tendrá el TP compilado y listo para probar. Asegúrese de tener instalado el **nasm**, el **gcc** y la **libsdl-gfx1.2-dev** para que esta acción se realice con éxito. Una vez que ya hizo lo antes mencionado, podríamos decir que usted ya tiene compiladas y linkeadas todas las partes del tp y generado el ejecutable que le permitirá correr el programa generado por nuestro código. Se generan 6 archivos .o correspondientes a los 6 archivos .asm (uno por cada función que tuvimos que realizar) y el archivo ejecutable que nos permite correr el programa. Como corresponde, este archivo se genera bajo el nombre de "runner".

El **Makefile** permite además correlo con el parámetro "clean", es decir "make clean" que borra todos los archivos generados al hacer el "make".

Para correr el programa, simplemente tipee "./runner" y presione enter. Una ventana se abrirá y podrá usted pasar un buen momento haciendo saltar, mover, trepar y golpear cubitos de monedas a **Mario**.

Si bien estamos en contra de que en el manual del usuario se expliquem los pasos a dar en

el mismo para ganarlo, creemos que, dado los tiempos veloces que corren en el siglo XXI, quizás no le alcance el tiempo al corrector para descubrirlo o para poder preguntarnos a nosotros, por lo cual le comentamos (no lea de aquí en más si no quiere saber la solución), que la solución... ¿está seguro de que quiere saberla y no descubrirla usted mismo?... es saltar muchas veces sobre el caño. Esta acción tiene un efecto extraño en **Mario** que lo hace saltar cada vez más alto, como si arriba del caño estuviera haciendo una dieta del Doctor Cormillot y a cada salto fuera más y más liviano. Este efecto se pierde y todo vuelve a la normalidad cuando **Mario** vuelve a pisar tierra firme o deja de saltar. Si **Mario** llega hasta el cielo, o parte superior de la pantalla, y colisiona con este, **Mario** "libera" a la **Princesa**. La **Princesa Durazno** aparece en la pantalla y el juego termina cuando **Mario** llega a saludarla.

Eso es todo lo que podemos informarle sobre nuestra versión del juego. Dudas, quejas, sugerencias a: grupo_orga_2@googlegroups.com . Si esta dirección no funciona le pedimos disculpas y le rogamos que reenvíe su mail a las direcciones de mail que figuran en la tapa de este informe.

Desde ya muchas gracias y esperemos que este juego e informe le hayan resultado amenos de leer y entretenido.

Plantel del grupo:

- Cristian
- Matías
- Tommy

¡Hasta la próxima!

5. Agradecimientos

No queremos terminar este informe sin antes agradecer a las siguientes personas que gracias a su compania, consejo y buena onda llegamos a terminar el tp que está usted leyendo en este momento

- Emi Höss
- Román
- Facu
- y a los mismos de siempre

¡Gracias totales!

Referencias

[INT97] Intel 64 and IA-32 Architectures Software Developer's Manual