Organización del Computador II

1er Cuatrimestre de 2008

Trabajo Práctico Nº 2:

"Rescatando a la Princesa Peach 2"
(la venganza de Homero)

Introducción

Cansado de saltar sobre sus oponentes, en esta nueva entrega de Güi, Mario decidió robarle un rayo *desmaterializado*r a los Cazafantasmas. Ya no tendrá que saltar más, solo apuntar y disparar para que todo desaparezca a su paso.

Además, Güi aprovechará esta segunda entrega para mejorar la performance de su *engine* con las tecnologías MMX y FPU.

Objetivo

En esta etapa del trabajo práctico se deberán implementar algunas funciones del TP1 utilizando el set de instrucciones de MMX y FPU de forma de optimizar los algoritmos del trabajo práctico anterior y de realizar ciertas operaciones aritméticas con números de punto flotante. También se agregarán algunas nuevas funcionalidades al juego.

Desarrollo

Para desarrollar este video juego, se debe programar en lenguaje ensamblador las siguientes funciones, respetando la convención C.

```
void generarFondo(char* screen, int ws, int hs,
                  char* piso, int wp, int hp);
```

Esta función debe ser reescrita utilizando las instrucciones MMX para el movimiento de los píxeles del color del fondo y del piso a la pantalla. En esta entrega el fondo de la pantalla debe ser de color negro, cuya codificación RGB es (0,0,0).

```
void recortar(char *imagen, int paso, int wp, int w, int h, char *res);
```

Esta función debe ser reescrita utilizando las instrucciones MMX para el movimiento de los píxeles del Sprite seleccionado al lugar apuntado por res.

```
void blit(char *sprite, int w, int h);
```

Esta función debe ser reescrita utilizando las instrucciones de MMX. Se recomienda utilizar una máscara que indique cuáles píxeles tienen color off (y por lo tanto deben ser reemplazados por el color del fondo). Se debe procesar por lo menos de a dos píxeles en paralelo.

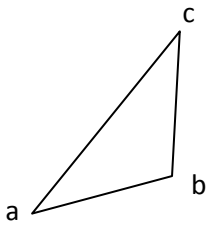
```
void chequearColisiones(short unsigned* puntos,
                        short unsigned longPuntos,
                        short unsigned x1, short unsigned y1, byte* res);
```

En esta etapa, utilizaremos la función **chequearColisiones** para chequear las colisiones entre un punto (x1,y1) y todos los Sprites en pantalla. Esta función recibe como parámetro un vector de números de 16 bits sin signo denominado **puntos** que contiene los cuatro vértices de cada Sprite (cada vértice del Sprite está representado con dos elementos del vector que indican sus coordenadas (x,y)). El parámetro **longPuntos** es la cantidad de elementos del vector **puntos**. Esta función recibe también dos números de 16 bits que representan las coordenadas del punto (**x1**,**y1**).

La función devuelve un vector de números de 8 bits denominado **res** de **longPuntos/8** elementos, donde el i-ésimo elemento indica si el punto colisiona con el i-ésimo Sprite (1 si colisiona, 0 si no).

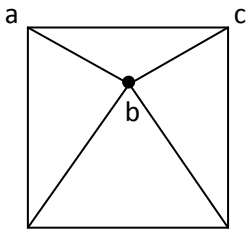
Para la implementación de esta función se debe considerar lo siguiente:

Utilizando la fórmula para calcular el área de un triangulo, se puede verificar si un punto está a la derecha o a la izquierda de un segmento. Veamos un ejemplo:



Si el área del triangulo *abc* es positiva, significa que *b* esta a la derecha del segmento *ac*, en cambio si es negativa significa que está a la izquierda.

Usando esto se pueden chequear colisiones dividiendo al rectángulo que representa al Sprite en cuatro triángulos, y calculando sus áreas. Todas las áreas resultan positivas si y sólo si el punto se encuentra dentro del rectángulo, es decir hay colisión.



Pero, ¿cómo calculamos el área de los triángulos? Una de las aplicaciones del producto vectorial es el cálculo de áreas de paralelogramos. De allí podemos deducir la siguiente fórmula para calcular el área de cada triángulo:

Área del triángulo abc= ½ [a1(b2-c2) + b1(c2-a2) + c1(a2-b2)]

donde (a1, a2) son las coordenadas del punto a; (b1, b2) son las coordenadas del punto b y (c1, c2) son las coordenadas del punto c.

Utilizar estos resultados para implementar la función con instrucciones de MMX.

```
void apagar(char *sprite, int w, int h, int *cont);
```

Esta función debe ser reescrita utilizando instrucciones de MMX

```
void salta(int *saltando, int *bajando, int*posy,
           int *velocidad, int *crecerVelocidad, int cotaAbajo);
```

En esta etapa del Trabajo Práctico, el sedentarismo ha llegado a Mario y en lugar de saltar opta por desmaterializar todo lo que encuentra a su paso. Por lo tanto esta función no se utiliza más.

```
void generarRayo( unsigned int xpI, unsigned int ypI ,
                  unsigned int xpF, unsigned int ypF ,
                  void* screen, unsigned int anchoPantalla);
```

Como Mario se cansó de saltar sobre sus oponentes, decidió armarse y le robó un rayo *desmaterializador* a los Cazafantasmas. Se debe implementar la función **generarRayo** que genera un rayo desde la posición (xpI, ypI), hasta la posición (xpF, ypF), siguiendo el siguiente pseudocódigo:

```
void generarRayo(xpI, ypI, xpF, ypF, screen, anchoPantalla);

angulo = anguloEntre( [(xpI , ypI), (xpI+10, ypI)],
                      [(xpF, ypF), (xpF, ypF+10)] );
x = 0;
mientras ( x < xpF)
{
      largo = 40;
      y = ( [largo * seno( ((x*2)*3.14/128)*2 )
            * cos ( ((x*2)*3.14/128)*8 )] / 5 );

      // Matriz de Rotación Inversa
      xf = cos(angulo)*x + seno(angulo)*y;
      yf = cos(angulo)*y – seno(angulo)*x;

      dibujar (xf, yf, screen, anchoPantalla);

      y = (       largo *
            seno( ((x mod 256)* 16 *3.14/128)*2 ) *
            cos ( ((x mod 256)*16 *3.14/128)*8 ) *
            seno( ((x mod 256)* 16 *3.14/128)*4 )  );

      // Matriz de Rotación Inversa
      xf = cos(angulo)*x + seno(angulo)*y;
      yf = cos(angulo)*y – seno(angulo)*x;

      dibujar (xf, yf, screen, anchoPantalla);
      x++;
}
```

Donde dibujar(x, y, screen, anchoPantalla) coloca un pixel de color a elección en la posición (x, y). Este color puede cambiar para cada llamada a la función, pero no se puede usar color negro ya que es el color del fondo.

Otras consideraciones

Para que puedan probar el trabajo práctico, les entregaremos un **`main`** realizado en lenguaje **`C`** que utiliza estas funciones, todas las imágenes que contienen los sprites a utilizar en este video juego y la librería **`SDL`** que es necesaria para la ejecución del mismo.

Informe

Se deben tener en cuenta los mismos puntos que en el Trabajo Práctico 1. Además se debe entregar los pseudocódigos de las funciones que se reescriban utilizando instrucciones de MMX y FPU para este trabajo práctico.

Entrega

La fecha de entrega de este trabajo es el **jueves 3 de julio** en el horario de clase (17 a 20 hs). No se aceptarán trabajos pasada esa fecha. Para poder contar con una instancia de recuperación del trabajo es necesario haber entregado una primera versión en la fecha antes mencionada. Recordamos que ese mismo día es la reentrega del Trabajo Práctico 1 para quienes no hayan aprobado.

La entrega se realizará en un CD que debe incluir, los ejecutables, todos los archivos fuentes necesarios para crearlos y el informe en pdf. Si desarrollaron prototipos en lenguaje C para resolver el trabajo primero en alto nivel, deben entregarlos e incluir los resultados obtenidos en el informe.