Organización del Computador II

1er Cuatrimestre de 2009

Trabajo Práctico N°1: “ Oportuncrisis ”

Introducción

La crisis llegó a los hogares del mundo y dejó a miles de personas sin la posibilidad de poder acceder a costosas placas de video para poder correr los últimos juegos del mercado.

Sin embargo, como mencionó un gran filósofo de la era contemporánea “crisis también significa oportunidad, ‘Oportuncrisis!’ ”, y tomando esta iniciativa un grupo de programadores argentinos comenzó a sacar al mercado títulos de juegos que corren puramente sobre el procesador de la computadora, sin utilizar los costosos beneficios de una placa de video que ya cada vez posee menos gente.

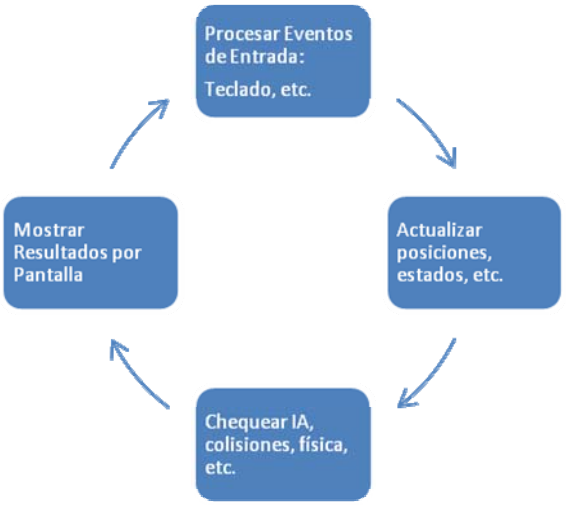
Por esto se pidió a un grupo de estudiantes de Ciencias de la Computación que utilicen sus altos conocimientos en Assembly para que logren usar todo el provecho posible que un procesador puede darle a una aplicación gráfica a través de la arquitectura de un Intel x86 y de las instrucciones de SSE.

Este Trabajo Práctico consistirá entonces en realizar, utilizando instrucciones de Assembly, la implementación de un modesto video juego.

Concepto

Para llevar a cabo esto veamos primero como será el funcionamiento:

Un video juego no es más que un programa que contiene los siguientes tres estados generales: la inicialización, el ciclo general, y la finalización. El primero es el encargado de inicializar la librería gráfica, el modo gráfico, reservar espacio para variables o estructuras que luego serán usadas, el tercero por su parte, será el encargado de liberar los recursos una vez que el juego termine, y el segundo, que es el que más nos importa, es un ciclo que cumple con el siguiente esquema:



Durante toda la ejecución del Video Juego, se ejecuta este ciclo una y otra vez, generalmente 30 o 40 veces por segundo, a cada ciclo se lo conoce con el nombre de Frame, y la cantidad de ciclos que realiza un juego por segundo se lo conoce como FPS (Frames Per Second), esto cicla hasta que por alguna razón se aborte y se pase al estado 3 en donde se finaliza y cierra el programa.

Para este Trabajo Práctico, nos vamos a concentrar principalmente en como mostrar resultados por pantalla, esto significa implementar en Assembly una serie de funciones y estructuras necesarias para que sea posible la visualización y animación de los personajes e ítems del juego, y el ambiente donde se encuentran los mismos.

Sprites y Animaciones

Un personaje en un Video Juego en 2 dimensiones muchas veces no es más que un dibujo que se muestra por pantalla repetidamente y a una velocidad tal que en algunas ocasiones genera la sensación de movimiento por parte del mismo, a este dibujo se lo llama Sprite y el siguiente es un ejemplo del mismo:

Para lograr entonces una animación utilizando un Sprite en 2D, simplemente es necesario recortar sistemáticamente una instancia del personaje para mostrarla en un Frame y luego recortar la instancia siguiente para mostrarla en el Frame siguiente, esta será una de las funciones que deberán hacer para el TP, la cual tiene el nombre de 'recortar' y se especifica a continuación.

```
void recortar(Uint8* sprite, Uint32 instancia, Uint32 ancho_instancia,
                    Uint32 alto_sprite, Uint32 ancho_sprite, bool orientacion)
```

Se deberá recortar la instancia que define el valor del último parámetro de la función, y pegarla en el buffer que viene en el parámetro 'sprite' orientada de acuerdo al último parámetro (true = original y false = invertida horizontalmente).

Veamos un ejemplo:

Esta imagen tiene 96 pixeles de ancho por 64 de alto, y contiene 3 Instancias, donde cada una tiene 32 pixeles de ancho. Entonces, si queremos recortar la instancia del medio, debemos invocar a la función con los siguientes parámetros:

```
recortar(&imagen, 1, 32, 96, 64, &res, true);
```

También es necesario mostrar un fondo para que el personaje no parezca estar flotando en medio de la nada, para dicho propósito se implementará una función llamada ‘generarFondo’ la cual a partir de una imagen para el fondo y una coordenada x en pixeles de la imagen, rellena todo el fondo de la pantalla con la imagen a partir de x.

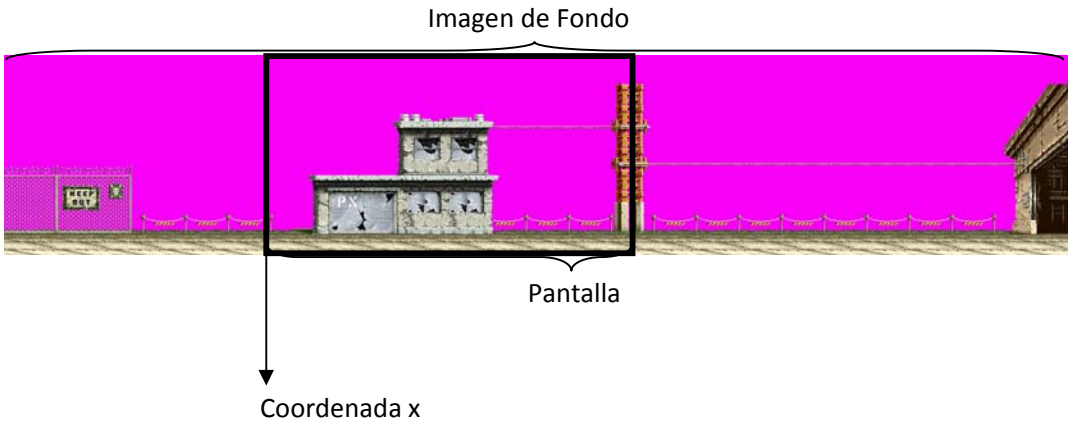
Es decir, si nuestra pantalla tiene 400 x 200 px y la imagen para dibujar de fondo tienen 800 x 200 px, si la coordenada x tiene el valor 20 significa que las primeras 19 columnas de pixeles de la imagen no hay que copiarlas, y hay que rellenar el fondo a partir de la columna 20 hasta la columna 420, que son las necesarias para cubrir la pantalla.

Si la coordenada a partir de donde se debe comenzar a copiar la imagen para el fondo es demasiado grande al punto de que no hay imagen suficiente para rellenar la pantalla, dicha coordenada deberá reemplazarse por el mayor valor que pueda tomar la misma; en el ejemplo anterior la mayor coordenada posible es 400.

La función es la siguiente:

```
void generarFondo(Uint8* fondo, Uint32 ancho_fondo, Uint32 alto_fondo,
                                  Uint32 coord)
```

Veamos un ejemplo:



Color Off

Tanto la imagen de fondo, como el sprite de un personaje tienen un color de fondo que no deberá mostrarse por pantalla, dicho color se lo denomina “color off” y sirve para que el personaje se vea más inmerso en la escena y no con un color de fondo que no concuerda con el escenario. Para esto se deberá implementar la función ‘blit’ que toma una imagen, un color-off y pega la imagen en la pantalla apropiadamente.

```
void blit(Uint8 *sprite, Uint32 ancho, Uint32 alto, Uint x, Uint y,
                        Color color-off);
```

Esta función es la encargada de solapar un sprite sobre el fondo de manera de que sólo se vea el dibujo, y no el color off. El parámetro ‘**sprite’** es un puntero a un área de memoria en donde está almacenado con dimensiones **ancho** y **alto , x** e **y** representan las coordenadas dentro de la pantalla de donde deben copiar el fondo para el sprite, el parámetro ‘colo-off’ representa el color que deberá solaparse con el fondo de manera apropiada para que detrás del personaje se vea el fondo que corresponde dependiendo de la posición donde se encuentre.

La coordenada de un personaje o ítem dentro de la pantalla representa el extremo inferior izquierdo del recuadro que ocupa.

Ejemplo de personajes sin blit:

Ejemplo de personajes con blit:

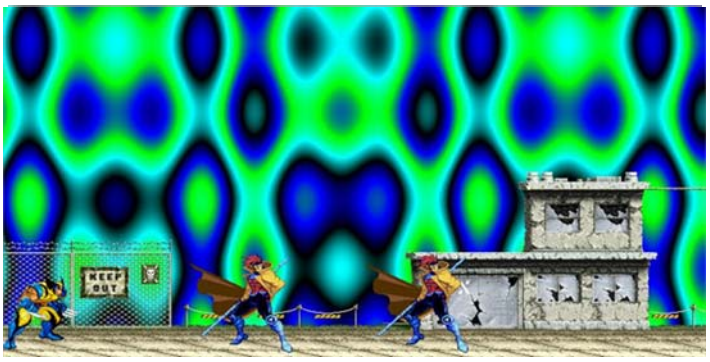
Efectos Visuales

Para rellenar el cielo del escenario, el cual en primera instancia se encuentra todo del mismo color, vamos a utilizar una función que genere un efecto visual denominado 'Plasma'.

El pseudocódigo de la función es el siguiente:

```
void generarPlasma (Color rgb)
{
   int x;
   for (i = 1; i <= SCREEN_HEIGHT; i++)
   {
      for (j = 1; j <= SCREEN_WIDTH; j++)
      {
         x =              colores[(g_ver0 + 5*j) mod 512] +
         colores[(g_ver1 + 3*j) mod 512] +
         colores[(g_hor0 + 3*i) mod 512] +
         colores[(g_hor1 + i) mod 512];

         index = 128 + (x >> 4);

         if (color_de_fondo[i, j] == rgb)
         {
            case index
               index < 64
                  color_de_fondo[i, j].r = 255 - ( (index << 2) + 1);
                  color_de_fondo[i, j].g = index << 2;
                  color_de_fondo[i, j].b = 0;
               64 <= index < 128
                  color_de_fondo[i, j].r = (index << 2) + 1;
                  color_de_fondo[i, j].g = 255;
                  color_de_fondo[i, j].b = 0;
               128 <= index < 192
                  color_de_fondo[i, j].r = 255 - ( (index << 2) + 1);
                  color_de_fondo[i, j].g = 255 - ( (index << 2) + 1);
                  color_de_fondo[i, j].b = 0;
               192 <= index < 256
                  color_de_fondo[i, j].r = (index << 2) + 1;
                  color_de_fondo[i, j].g = 0;
                  color_de_fondo[i, j].b = 0;
               index >= 256
                  color_de_fondo[i, j].r = 0;
                  color_de_fondo[i, j].g = 0;
                  color_de_fondo[i, j].b = 0;
         }

      }
   }
   g_ver0 += 9;
   g_hor0 += 8;
}

Las variables g_ver0, g_ver1, g_hor0, g_hor1 son variables globales.
```

Para tener una idea de cómo debería verse el fondo agregamos una captura del juego ya realizado en donde se puede ver el cielo generado por este efecto.



A su vez, cada vez que el personaje principal de nuestro juego llega al final del primer escenario, se produce un efecto conocido como ‘Smooth’, donde los pixeles poco a poco van apagándose mediante una función que definiremos a continuación, produciendo que la pantalla vaya oscureciéndose salvo nuestro personaje, en un momento determinado cuando la mayoría de los pixeles están en negro, la función devuelve un valor booleano ‘true’ para informar que el efecto terminó de ‘apagar’ la pantalla.

El pseudocódigo de dicha función es el siguiente:

```
bool smooth()
{
        Para todo color c de todo pixel de la pantalla
        {
                sumatoria = superior(c) + inferior(c) + sig(c) + ant(c);
                p = ( sumatoria )/4;

                if (sumatoria == 0)
                        contadorNegros++;
                else
                        contadorBlancos++;
        }
        return contadorNegros > contadorBlancos;
}
```

superior(c): Devuelve el valor del color c que se encuentra en el pixel superior al pixel donde se encuentra, si el mismo no existe devuelve cero.

inferior(c), sig(c), ant(c) : Idem, pero con el pixel inferior, siguiente y anterior respectivamente.

Función Negativo

Cada vez que se presione la tecla ‘v’ se producirá un efecto en el juego similar al negativo de una foto pero con colores y durará el efecto mientras se mantenga apretada la tecla.

Deberán entonces implementar este efecto a partir del siguiente pseudocódigo utilizando instrucciones de SSE para obtener resultados en punto flotante, la elección de utilizar 32 o 64 bits para representar los números queda a su decisión.

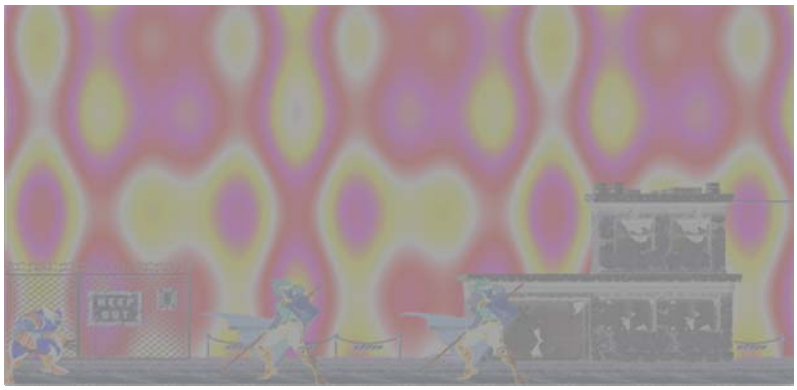
El pseudocódigo es el siguiente:

```
bool negative()
{
        Para todo color c de todo pixel de la pantalla
        {
                sumatoria = superior(c) + inferior(c) + sig(c) + ant(c) + 1;
                p = (1 / sumatoria) * 255;

        }
}
```

superior(c): Devuelve el valor del color c que se encuentra en el pixel superior al pixel donde se encuentra divido el mayor número representable en 8 bits sin signo, si el pixel no existe devuelve cero.

inferior(c), sig(c), ant(c) : Idem, pero con el pixel inferior, siguiente y anterior respectivamente.

En la siguiente captura de pantalla puede verse como debería visualizarse el efecto:



Listas de Items

Además del personaje principal, también existen otros personajes secundarios e ítems que aparecen en el juego en una posición en particular y no se mueven demasiado.

Para almacenar dichos ítems y sus posiciones deberán implementar una lista con su respectivo iterador, en donde figurarán los ítems y/o personajes secundarios que se encuentren activos en el juego, dicha lista será consultada constantemente en cada Frame del juego para saber que se debe dibujar en pantalla. Las estructuras y funciones que deberán implementarse para dicho fin son las siguientes:

Estructuras para la lista:

```
struct Nodo
{
        Uint64      ID;
        SDL_Surface         *surfaceGen;
        SDL_Surface *surfacePers;

        Uint32      coord_x;
        Uint32      coord_y;
        Nodo*       prox;
        Nodo*       prev;
};

struct Lista
{
        Nodo* primero;
};
```

Funciones para implementar de la Lista:

Lista* constructor_lista() : Construye una lista, pidiendo memoria para el puntero al primer nodo, y lo inicializa en null y devuelve el puntero.

bool verificar_id(Lista lis, Uint32 id) : Recorre la lista lis verificando que ningún elemento tenga ID = id, en ese caso devuelve true, sino false.

void agregar_item_ordenado (Lista* la_lista, SDL_Surface* surfacePers, SDL_Surface* surfaceGen, Uint32 x, Uint32 y, Uint32 ID) : Agrega un elemento a la lista ordenándolo de acuerdo a la coordenada x del elemento.

void borrar(Lista* la_lista, Uint32 x, Uint32 y) : Borra todo elemento de la lista que este encerrado entre las coordenadas x e y con un rango de 50 pixeles.

void liberar_lista(Lista l) : Libera el espacio de memoria de la lista.

Estructuras para el Iterador:

```
struct Iterador
{
        Nodo *actual;
};
```

Funciones para implementar del Iterador:

Iterador* constructor_iterador(Lista *lista) : Guarda espacio para un iterador y apunta a lo mismo que apunta lista.

void proximo(Iterador *iter) : Avanza el iterador de la lista al próximo elemento de la lista, si este existe

Nodo* item(Iterador *iter) : Devuelve el nodo de la lista apuntado por el iterador.

bool hay_proximo(Iterador *iter) : Verifica si hay un elemento siguiente al actual apuntado por el iterador.

void liberar_iterador(Iterador iter) : Libera el espacio apuntado por el iterador:

Instancias del Trabajo Práctico

La entrega de este Trabajo Práctico consta de dos instancias.

Para la primera etapa deberán implementarse todas las funcionalidades anteriormente detalladas en Assembly, salvo smooth y negative, utilizando únicamente registros de propósito general, variables locales y/o globales y constantes. Pueden usarse las instrucciones dadas en clase y las que figuren en el manual de Intel que no necesiten utilizar otros operandos más que los autorizados para la primera entrega.

Para la segunda etapa del Trabajo Práctico, deberán re-implementar las funciones de recortar, blitear, generarFondo, generarPlasma, e implementar smooth y negative utilizando la arquitectura de SSE y desarrollando una metodología de programación nueva que logre aprovechar y optimizar los algoritmos utilizando dicha arquitectura. Toda optimización que se haga deberá ser documentada y explicada en el informe.

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.

Para trabajar con imágenes es necesario tomar en cuenta que son BMP’s y tratar a toda imagen del juego con las mismas características con las que se trataría un BMP salvo que los pixeles están ordenados de izquierda a derecha y de abajo a arriba igual que como se

los ve en pantalla. El fondo de la pantalla no es más que una matriz de pixeles de 24 bits y es lo único que no debe considerarse como un BMP.

Informe y forma de entrega

El informe debe reflejar todo el trabajo realizado, las decisiones tomadas (con el estudio de sus alternativas), las estructuras de datos usadas, el testing que hayan hecho para detectar errores y optimizar código y los resultados obtenidos. Debe contar como mínimo con los siguientes capítulos: introducción, desarrollo, resultados y conclusiones. Debe estar estructurado top-down, es decir, leyendo la introducción se debe saber qué se hizo y cuáles son las partes más importantes. En el capítulo de desarrollo se deben detallar las decisiones que se tomaron, las estructuras de datos que se utilizaron y la implementación de cada una de las funciones programadas en lenguaje ensamblador. El capítulo de resultados debe contener un análisis tanto en los casos exitosos como en lo que no se obtuvieron los resultados esperados. Por último, se presentan las conclusiones del trabajo.

Además, el informe debe incluir una carátula con nombre del grupo (tiene que ser alguna instrucción del set de instrucciones de Intel) y de los integrantes con número de libreta y email, instrucciones para el corrector y el detalle de todos los archivos entregados. La fecha de entrega de este trabajo es el martes 28 de abril para la primera parte y el martes 19 de mayo para la segunda (en el horario de clase de 17 a 22 hs). No se aceptarán trabajos pasada esa fecha. Si un grupo decide no entregar nada en la primera fecha, va directo a recuperatorio (para la primera parte el recuperatorio se entrega junto con la segunda parte y para la segunda, el recuperatorio se entrega en la fecha de entre del segundo trabajo práctico). No hay segundo recuperatorio. Por eso conviene fuertemente que entreguen en primera fecha lo que tengan hecho hasta el momento para que les corrijamos esa parte. Documenten a medida que realizan el trabajo, no dejen esto para el final. Se pide entregar una carpeta con el informe del trabajo impreso y un CD con los siguientes directorios:

- src: contiene los archivos fuente.
- exe: contiene los archivos ejecutables.
- enunciado: contiene este documento.
- informe: contiene el informe en formato .pdf.