Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación

Proyecto 1 - ScreenSaver

Andres Paiz 191142
Oscar Ramos 19184
Rene Ventura 19202

Guatemala, 5 de abril de 2023

Introducción

En el mundo de la informática, el desempeño de los programas es uno de los factores más importantes que determinan su efectividad. En este informe se comparará el desempeño de dos programas de protección de pantalla: uno secuencial y otro paralelo.

El programa secuencial funciona a través de una serie de instrucciones que se ejecutan una tras otra. Por otro lado, el programa paralelo divide las tareas en pequeñas unidades y las ejecuta simultáneamente en múltiples núcleos del procesador, aprovechando al máximo la capacidad de la CPU.

El objetivo de este informe es comparar el rendimiento de estos dos programas de protección de pantalla en términos de velocidad y eficiencia. Para ello, se llevaron a cabo pruebas utilizando diferentes configuraciones de hardware y software, y se registraron los resultados obtenidos.

Antecedentes

Los screensavers son programas que se ejecutan en un equipo después de que el ratón y el teclado hayan estado ociosos durante un período de tiempo específico. Los desarrolladores crean módulos de código fuente que contienen tres funciones requeridas y las enlazan con la librería de screensaver para crear efectos visuales. Además, un screensaver debería suplir un ícono que sirve como herramienta de marketing y publicidad para una empresa.

Para la creación de screensavers es importante tener en cuenta conceptos y teorías de la física clásica como la cinética y cinemática de cuerpos sólidos y la generación de partículas. También es crucial la matemática implicada en la detección de colisiones de objetos, así como conceptos como matrices, cuaterniones y vectores necesarios para calcular rotaciones, traslaciones, fuerzas y otros elementos requeridos en la simulación o creación de un screensaver o incluso videojuegos.

Es importante destacar que la física de los videojuegos, a menudo, no requiere una gran precisión y se simplifican los escenarios para lograr mayor velocidad de procesamiento. Los screensavers también pueden ser utilizados como una manera de difundir la imagen de una empresa y acercarse más a sus clientes. En resumen, los screensavers no solo son una herramienta útil para proteger una pantalla de quemaduras de fósforo y ocultar información delicada, sino también una oportunidad para la creación de efectos visuales y publicidad.

Cuerpo

Se utilizaron dos librerías principales para el desarrollo de este proyecto, SDL2 es una biblioteca de gráficos de alta calidad que proporciona a los desarrolladores una plataforma multiplataforma para la creación de aplicaciones gráficas y multimedia en C/C++. Ofrece funciones para la creación de ventanas, dibujo en pantalla, entrada de teclado y mouse, sonido, entre otras características. También proporciona soporte para la aceleración de hardware en gráficos 2D y 3D.

Por otro lado, SDL2_ttf es una biblioteca que proporciona soporte para fuentes TrueType (TTF) en SDL2. Permite a los desarrolladores cargar fuentes TTF para ser utilizadas en aplicaciones gráficas y multimedia, proporcionando una calidad de fuente mejorada en comparación con las fuentes predefinidas. Esto es particularmente útil para la creación de interfaces de usuario con texto de alta calidad y legibilidad.

Ambas bibliotecas son gratuitas y de código abierto, lo que las hace ideales para el desarrollo de proyectos de software libre y comercial. Además, proporcionan una documentación completa y ejemplos de código que facilitan su implementación en los proyectos.

OpenMP es una interfaz de programación de aplicaciones (API) de multiprocesamiento en memoria compartida para sistemas multinúcleo y multiprocesador. Permite a los programadores escribir programas que utilizan múltiples hilos de ejecución concurrentes, que pueden ejecutarse simultáneamente en múltiples núcleos de CPU, mejorando así el rendimiento de la aplicación. OpenMP proporciona un conjunto de directivas de compilador, rutinas de biblioteca y variables de entorno que permiten al programador especificar la forma en que se deben dividir y ejecutar los bucles y otros fragmentos de código en múltiples hilos. OpenMP es compatible con C, C++ y Fortran. Es ampliamente utilizado en aplicaciones científicas y de ingeniería que requieren cálculo intensivo y procesamiento en paralelo.

Para la paralelización se definieron 16 threads, además de modificar la estructura del programa secuencial. En los cambios de la modificación se atomizó el codigo, se eliminaron condiciones innecesarias para la paralelización como el chequeo doble de colisión de las estrellas además de utilizar las directivas (**#pragma omp parallel for y #pragma omp parallel for schedule(dynamic)**que se agregaron para paralelizar los for loops para actualizar la posición de las estrellas y manejar las colisiones de estrellas. La paralelización es una técnica para dividir una tarea en subtareas más pequeñas que pueden ser ejecutadas simultáneamente por múltiples threads, lo que puede mejorar el rendimiento y reducir el tiempo de ejecución del programa, especialmente para tareas computacionalmente intensivas que pueden se paralelizar fácilmente.

En este caso, los for loops se utilizan para actualizar la posición de las estrellas y manejar las colisiones de estrellas ya que implican iteraciones independientes que se pueden ejecutar en paralelo, lo que los convierte en buenos candidatos para la paralelización. La directiva **#pragma omp parallel for** crea un equipo de threads que ejecutan las iteraciones del for en paralelo, y la directiva **#pragma omp parallel for schedule(dynamic)** divide las iteraciones del for en fragmentos que se asignan dinámicamente a los threads, lo que puede mejorar el load balancing y reducir la sobrecarga de crear y sincronizar threads.

```
int max_threads= 16;
 omp_set_num_threads(max_threads);
```

```
if (stars[i].bounces > 10 || stars[i].radius <= 0) {
    // limpia el array de estrellas
    #pragma omp parallel for
    for (int j = i; j < num_stars - 1; j++) {
        stars[j] = stars[j + 1];
    }
    num_stars--;
```

```
// Handle star collisions
#pragma omp parallel for schedule(dynamic)
for (int i = 0; i < num_stars; i++) {
    for (int j = i + 1; j < num_stars; j++) {
            Uint32 elapsed_time_collision_i = SDL_GetTicks() - stars[i].start_time_collision;
            Uint32 elapsed_time_collision_j = SDL_GetTicks() - stars[j].start_time_collision;
            bool ignoreCollision_i = elapsed_time_collision_i < 5000;
            bool ignoreCollision_j = elapsed_time_collision_j < 5000;
            bool ignoreCollision = ignoreCollision_i || ignoreCollision_j;

            if (isCollision(stars[i], stars[j], ignoreCollision)) {

                ballCollisionManager(stars[i], stars[j]);

            }
    }
}
```

Conclusiones

- El desarrollo de un screensaver es una tarea interesante que combina conocimientos de programación y física para poder crear efectos visuales atractivos y eficientes.

- Las librerías como SDL2 y libsdl2-ttf son herramientas útiles para facilitar el proceso de programación de un screensaver.

- El uso de paralelismo proporcionó una mejora en el rendimiento del screensaver.

Recomendaciones

- Se recomienda utilizar linux como sistema operativo para así ahorrar tiempo en la configuración de las diferentes librerías que se utilizaran.

- A la hora de escribir el programa secuencial es útil marcar de alguna forma las secciones que podemos paralelizar.

- Para la comprensión de las librerías al igual que el funcionamiento de OpenMP, utilizar las fuentes de información oficiales fue de gran ayuda y son recomendables.

- Mejorar el rendimiento del screensaver: Al utilizar tecnologías como OpenMP para la ejecución paralela, se puede mejorar significativamente el rendimiento del screensaver. Sin embargo, es importante asegurarse de que el código esté optimizado y bien estructurado para evitar problemas de rendimiento o errores. Se recomienda realizar pruebas exhaustivas para garantizar que el screensaver funcione sin problemas.

- Agregar opciones de personalización: Una forma de mejorar la experiencia del usuario es permitir que pueda personalizar el screensaver. Se pueden agregar opciones para elegir la velocidad de movimiento de las figuras, el tipo de figuras a mostrar, el color de fondo, entre otros.

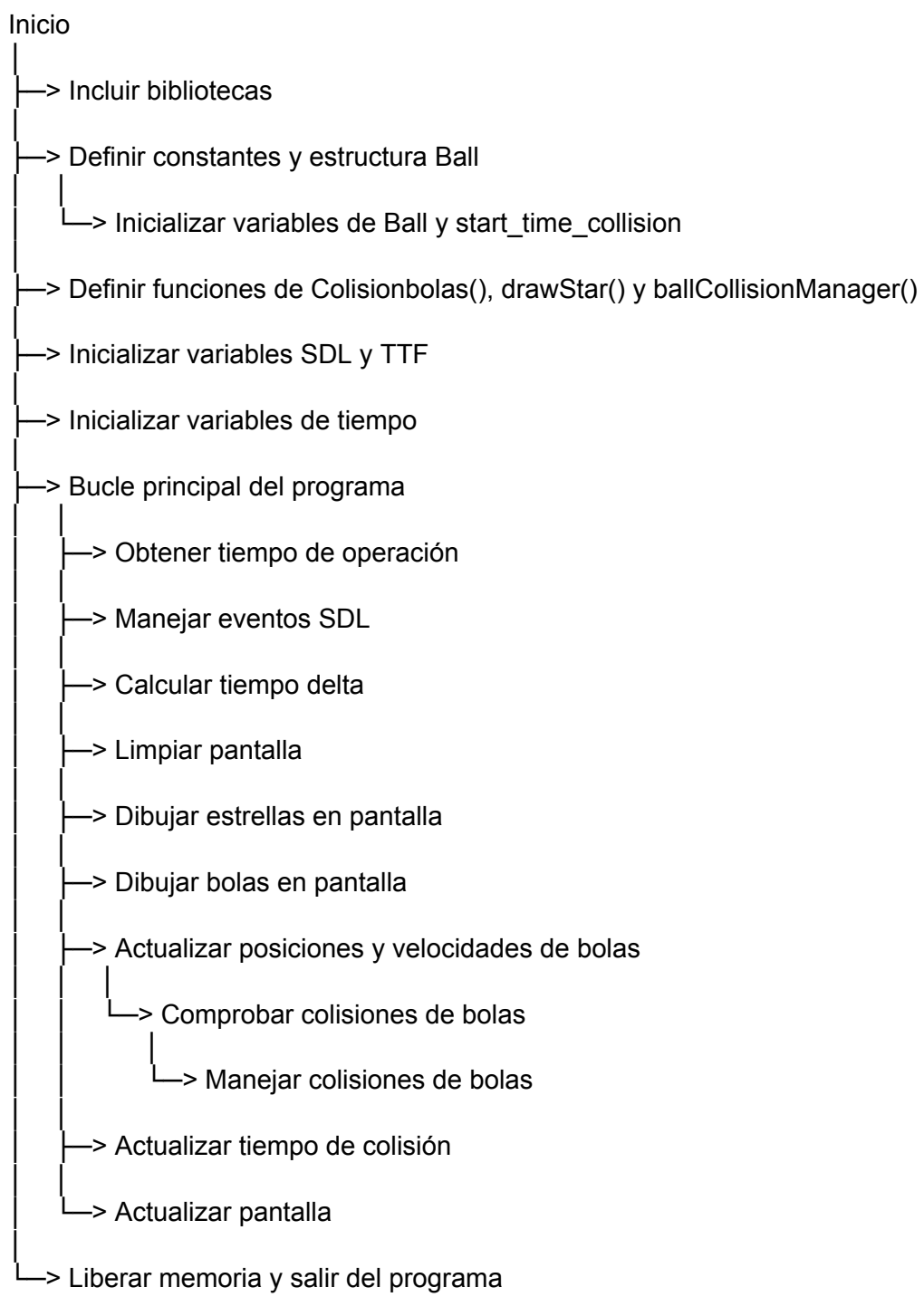
- Considerar la compatibilidad con diferentes sistemas operativos: Asegurarse de que el screensaver sea compatible con diferentes sistemas operativos y configuraciones de hardware puede ampliar su alcance y aumentar su utilidad. Se recomienda realizar pruebas en diferentes sistemas operativos y configuraciones para garantizar que el screensaver funcione correctamente en todos los casos.

Apéndice

Existen dos archivos, uno conteniendo el programa secuencial y el otro paralelo.

Anexo 1

Diagrama de flujo de programa paralelo



Anexo 2

Catálogo de funciones

- Colisionbolas: Esta función determina si dos bolas han colisionado. Toma dos objetos Ball como parámetros de entrada y devuelve true si las dos bolas han colisionado y false en caso contrario.

- drawStar: Esta función dibuja una estrella en la pantalla. Toma como parámetros de entrada el renderizador de SDL, las coordenadas x e y del centro de la estrella y el radio de la estrella.

- ballCollisionManager: Esta función gestiona la colisión entre dos bolas. Toma dos objetos Ball como parámetros de entrada y actualiza sus posiciones y velocidades en función de la física de la colisión.

- main: Esta es la función principal del programa. Crea la ventana de SDL, inicializa las bolas, dibuja las bolas y las estrellas en la pantalla y maneja las colisiones entre las bolas.

- Cabe mencionar que también hay algunas variables definidas a nivel global, como sheight, swidth, bradius, grav, bounce, bcol, start_time_operations, end_time_operations, total_time_operations, start_time_collision y Collision, pero estas no son funciones en sí mismas.

Anexo 3

Para 3000 estrellas generadas

secuencial(ms)	paralelo(ms)
837	204
784	186
822	182
819	195
835	189
832	187
942	180
945	189
990	189
1037	192
844	198
921	209
994	199
955	203
1138	180
1102	193
1031	184
speedup	**4.856704511**
efficiency	**0.303544032**

```
Ingrese el numero de estrellas tiene que ser mayor a 0: 3000
Tiempo total de ejecucion 204 ms
Tiempo total de ejecucion 186 ms
Tiempo total de ejecucion 182 ms
Tiempo total de ejecucion 195 ms
Tiempo total de ejecucion 189 ms
Tiempo total de ejecucion 187 ms
Tiempo total de ejecucion 180 ms
Tiempo total de ejecucion 189 ms
Tiempo total de ejecucion 189 ms
Tiempo total de ejecucion 192 ms
Tiempo total de ejecucion 198 ms
Tiempo total de ejecucion 209 ms
Tiempo total de ejecucion 199 ms
Tiempo total de ejecucion 203 ms
Tiempo total de ejecucion 180 ms
Tiempo total de ejecucion 193 ms
Tiempo total de ejecucion 192 ms
Tiempo total de ejecucion 185 ms
Tiempo total de ejecucion 184 ms
Tiempo total de ejecucion 180 ms
Tiempo total de ejecucion 191 ms
Tiempo total de ejecucion 185 ms
^CTiempo total de ejecucion 184 ms
Number of threads: 16
Promedio de FPS: 1.79058
```

```
Ingrese el numero de estrellas tiene que ser mayor a 0: 3000
Tiempo total de ejecucion 837 ms
Tiempo total de ejecucion 784 ms
Tiempo total de ejecucion 822 ms
Tiempo total de ejecucion 819 ms
Tiempo total de ejecucion 835 ms
Tiempo total de ejecucion 832 ms
Tiempo total de ejecucion 946 ms
Tiempo total de ejecucion 945 ms
Tiempo total de ejecucion 990 ms
Tiempo total de ejecucion 1037 ms
Tiempo total de ejecucion 844 ms
Tiempo total de ejecucion 921 ms
Tiempo total de ejecucion 994 ms
Tiempo total de ejecucion 955 ms
Tiempo total de ejecucion 1138 ms
Tiempo total de ejecucion 1102 ms
Tiempo total de ejecucion 1031 ms
^CTiempo total de ejecucion 1083 ms
Tiempo total de ejecucion 1185 ms
Promedio de FPS: 0.75664
```

Referencias

SDL2/Frontpage (no date) *SDL2/FrontPage - SDL Wiki*. Available at: https://wiki.libsdl.org/SDL2/FrontPage (Accessed: April 10, 2023).

OpenMP (no date). Available at: https://www.openmp.org/wp-content/uploads/OpenMPRefCard-5-2-web.pdf (Accessed: April 10, 2023).

Chapter 6 performance considerations (no date) *Chapter 6 Performance Considerations (Sun Studio 12: OpenMP API User's Guide)*. Available at: https://docs.oracle.com/cd/E19205-01/819-5270/6n7c71vec/index.html (Accessed: April 10, 2023).