



Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC3069 Computación Paralela y Distribuida
Catedrático: Miguel Novella
Ciclo 1 de 2023

Proyecto 1: Screensaver

Guido Padilla 19200
Oscar Saravia 19322
Oscar Paredez 19109

Guatemala, 22 de marzo de 2023

Índice

Índice	1
Introducción	2
Antecedentes	2
Desarrollo	3
Conclusiones	8
Recomendaciones	8
Apéndice	8
Instalación de un compilador de C en Windows	8
Instalación de OpenGL en Windows	9
Bibliografía	9
Anexos	10
Diagrama de flujo	11
Instrucciones para correr los programas	12
Catálogo de Funciones	12
Bitácora de Pruebas	15
Cálculo de speedup y efficiency	15
Capturas de los tiempos obtenidos en la ejecución del programa	16

Introducción

Un screensaver (también conocido como protector de pantalla) es un programa que se activa automáticamente después de cierto tiempo sin actividad en la mayoría de dispositivos electrónicos, como en una computadora o en una televisión (Tatum, 2023). Su objetivo principal es representar períodos largos de inactividad en el dispositivo o el bloqueo del mismo, mostrando imágenes en movimiento o patrones. Además, los screensavers también pueden ser utilizados con fines estéticos o de entretenimiento, ya que pueden mostrar animaciones, una secuencia de fotografías, información útil o simplemente imágenes atractivas. En resumen, los screensavers proporcionan una experiencia visual atractiva cuando no lo estás usando. El objetivo de este proyecto es desarrollar esta experiencia visual aplicando el concepto de paralelización. El fin de esto último es mantener un rendimiento alto mediante distribución de tareas y alcanzar mayor capacidad de respuesta, aplicando los conocimientos adquiridos en el curso hasta entonces.

Antecedentes

OpenMP es una librería bastante famosa que como se mencionó anteriormente, ayuda a optimizar el rendimiento y la eficiencia de un programa para que sea menos complicado para el sistema computacional de procesar lo requerido. Una diversidad de empresas han decidido implementar OpenMP en sus proyectos de desarrollo con el fin de, desde el punto de vista del usuario y su experiencia, brindar una mejor interacción con el proyecto en mención (Chapman et al., 2008).

Por ejemplo, Google ha decidido implementar paralelismo en algunos de sus proyectos, como por ejemplo su motor de búsqueda, Google Chrome, entre otras menos conocidas. Esto fue implementado con el fin de permitir a los usuarios llevar a cabo tareas complejas e intensas computacionalmente de una manera más eficiente.

Asimismo, una empresa que tiene involucramiento en muchas de las computadoras de hoy en día. Como todos sabemos, NVIDIA es una empresa que básicamente se enfoca en la paralelización dentro de sus productos, los cuales son las tarjetas gráficas, también llamadas GPUs. Mediante esta técnica, se mejora la eficiencia con la cual se mejoran los gráficos visualizados en una pantalla, incluyendo CUDA y su propia biblioteca de paralelización (Heller, 2022).

Existe otra organización que desarrolló un proyecto llamado Weather Research and Forecasting Model. Este proyecto tiene como principal objetivo simular la dinámica atmosférica implementando modelos matemáticos computacionalmente pesados.

Ellos decidieron implementar no solo paralelización, sino que también la librería de OpenMP con el fin de agilizar los cálculos en diferentes núcleos de CPU. Con esto se alcanza un mejor rendimiento, así como unos cálculos más precisos a lo largo del procedimiento matemático realizado (Powers et al., 2017).

La elaboración de screensavers utilizando programación paralela implica el uso de múltiples procesadores o núcleos de CPU para realizar tareas simultáneamente. Los screensavers son programas que se ejecutan cuando el sistema se encuentra en estado de inactividad y tienen la capacidad de mostrar animaciones, imágenes o efectos visuales (Markoff, 1990).

Dentro de proyectos donde se realizó un screensaver que utiliza programación paralela, podemos encontrar Parallel Screensaver, este proyecto es una implementación en C++ de un screensaver que utiliza múltiples hilos para dibujar patrones en la pantalla. Se utiliza la biblioteca OpenMP para la programación paralela.

Desarrollo

La temática del screensaver a desarrollar es representar una serie de esferas que tienen el “poder”, que en este caso las llamaremos **esferas influencers**, de atraer y una serie de esferas que son atraídas por las esferas influencers, las cuales llamaremos **esferas followers**. Todas las esferas inician con una velocidad bastante discreta y con una dirección de movimiento aleatoria. Se busca mediante una serie de conceptos de la física y de la matemática que las esferas followers estén calculando constantemente la distancia que tiene cada una de ellas con las esferas influencers. Debido a que todas se encuentran en movimiento, llegará un momento para todas las esfera followers que se encontrarán lo suficientemente cerca a una esfera influencer. Cuando llega este momento, se busca mediante la función de velocidad, que la esfera follower tome una velocidad mínimamente menor a la esfera influencer, con el fin de que la esfera influencer atraiga a la esfera follower. Debido a que la velocidad de la esfera follower es menor a la de la esfera influencer, llegará un punto en el que la distancia entre ambas esferas sea tanta que la esfera follower dejará de seguir a la esfera influencer.

Para desarrollar un screensaver desde 0 en C++, se hizo uso de cuatro librerías. La primera utilizada fue “iostream” la cual no es más que un facilitador para llevar el control de la lectura y escritura de en los distintos flujos. A continuación, se incluyó la librería “math”, la cual como su nombre indica, provee varias funciones matemáticas, con el fin de ahorrarnos cálculos que pueden llegar a ser algo complejos. En este caso, se utilizaron las funciones de seno y coseno para calcular, a grandes rasgos, la posición en X y Y para las esferas empleadas en el screensaver. Posteriormente, y probablemente la más importante, se utilizó la

librería de GLU, la cual provee funciones para simplificar la programación en OpenGL, como por ejemplo interacciones con el mouse y el teclado, la creación de una ventana de imagen, entre otras. A continuación, se importó `stdio.h` con el fin de tener la flexibilidad de obtener información de input y mostrar información como output. Finalmente, la librería de “vector” no es nada más que la implementación de la estructura de dato conocida como vector.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains six lines of C++ code, each starting with a line number from 1 to 6.

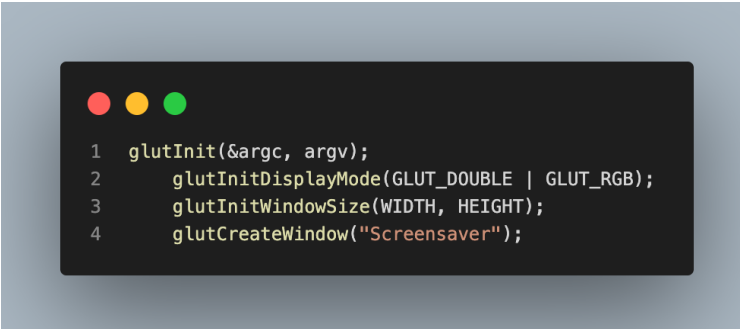
```
1  #include <iostream>
2  #include <math.h>
3  #include <GL/glut.h>
4  #include <omp.h>
5  #include <vector>
6  #include <stdio.h>
```

Teniendo en cuenta las librerías empleadas, lo primero que se realiza tras compilar el programa, es al correrlo, definir la cantidad de partículas que atraen y la cantidad de partículas que son atraídas. A continuación, definimos las posiciones iniciales tanto en el eje horizontal como en el vertical para cada una de las esferas, así como sus velocidades en ambos ejes.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains five lines of C++ code, each starting with a line number from 1 to 5.

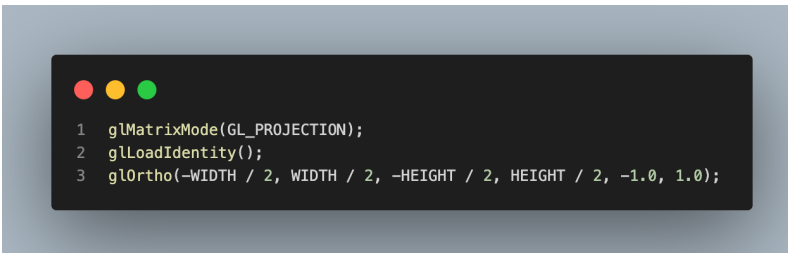
```
1  vector<bool> alfaParticle;
2  vector<float> arrayVX;
3  vector<float> arrayVY;
4  vector<float> arrayY;
5  vector<float> arrayX;
```

Ya teniendo definidos los parámetros iniciales para llevar a cabo la ejecución del screensaver, se procede a crear una nueva ventana, la cual va a ser donde vamos a desplegar nuestras animaciones, mediante la función de GLU **glutInit**. Posterior a eso, se define el modo de visualización y el tamaño de la ventana, mediante las funciones de **glutInitDisplayMode** y **glutInitWindowSize** respectivamente. Finalmente se le asigna un nombre a la ventana mediante **glutCreateWindow**.

A screenshot of a code editor window with a dark background and light-colored text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The code is as follows:

```
1  glutInit(&argc, argv);
2      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
3      glutInitWindowSize(WIDTH, HEIGHT);
4      glutCreateWindow("Screensaver");
```

Después de establecer los distintos parámetros que va a tener la ventana, se procede a establecer el modo de la matriz de transformación que va a tener nuestra ventana, la cual en nuestro caso fue matriz de proyección, que es la matriz en la cual vamos a representar nuestra animación. Esto se hace en la función de **glMatrixMode**. Luego de esto, se debe de cargar dicha matriz para establecerla como la matriz actual, es decir, la activa. Esto se realiza mediante **glLoadIdentity**. Como punto final de la preparación de la ventana, le asignamos un fondo de color negro mediante **glClearColor**.

A screenshot of a code editor window with a dark background and light-colored text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The code is as follows:

```
1  glMatrixMode(GL_PROJECTION);
2  glLoadIdentity();
3  glOrtho(-WIDTH / 2, WIDTH / 2, -HEIGHT / 2, HEIGHT / 2, -1.0, 1.0);
```

A estas alturas, ya se tiene configurada la ventana, así como la matriz sobre la cual se va a desarrollar la animación. Como primer punto, se llama la función **glutDisplayFunc** la cual recibe como parámetro la función de renderizado que se requiere cada vez que se necesite volver a dibujar la animación. Posterior a esto, se llama la función **glutTimerFunc** la cual en este caso se utilizó para llamar cierta función, en este caso una desarrollada por nosotros llamada **update**, cada cierto

tiempo. Se colocó un timer de 16 milisegundos con el fin de que se esté llamando prácticamente en todo momento esta última función mencionada. Como último paso, se inicia el bucle de eventos de GLUT mediante **GlutMainLoop**. Esta función nos permite que la animación y la ventana se mantengan activas hasta que el usuario cierre manualmente la ventana o termine el programa de alguna manera.

Finalmente, es necesario desarrollar sobre las dos funciones implementadas por el grupo. Estas son las funciones de **display** y **update**.

En la función de **update** es donde se realiza la implementación de tres conceptos de la física. Uno de ellos es la distancia entre dos puntos sobre un plano en dos dimensiones. Este concepto fue necesario implementarlo para saber qué tan cerca se encuentra cada una de las esferas followers de cualquier esfera influencer. Igualmente, se implementó el concepto de velocidad, para que cuando una esfera follower en movimiento pase cerca de una esfera influencer en movimiento, la esfera follower tome la dirección de la esfera follow y una velocidad mínimamente menor a la de la esfera influencer para que parezca que la está siguiendo. Otro concepto implementado es la aceleración. Esta se puede ver reflejada cuando una esfera follower se encuentra cerca de una esfera influencer. De ser así, la esfera follower, como se mencionó antes, toma la dirección y una velocidad mínimamente menor para dar la sensación de que la está siguiendo. Sin embargo, adicional a eso, la esfera follower toma una aceleración para acercarse lo más rápido posible a la esfera influencer. Se busca no excederse con dicha aceleración para que la animación sea amigable para la vista. Asimismo, se utiliza el concepto de aceleración para cuando una esfera follower va llegando a la posición de una esfera influencer ya que se busca desacelerar la esfera follower por lo mencionado anteriormente de proveer una animación agradable a la vista.

En la función de **display** a grandes rasgos no se hace nada más que reflejar los cálculos realizados en la función de **update**, información que se guarda en variables globales para que **display** tenga acceso a ellas. En resumen, en **update** se van calculando los distintos factores que afectan a cada una de las esferas, y en **display** se toman todos esos valores para representar las distintas animaciones en pantalla.

```

1 void display()
2 {
3     glClear(GL_COLOR_BUFFER_BIT);
4     for (int j = 0; j < alfaParticle.size(); j++)
5     {
6         if (alfaParticle.at(j))
7             glColor3f(0.0f, 0.0f, 1.0f);
8         else
9             glColor3f(1.0f, 0.0f, 0.0f);
10        glBegin(GL_TRIANGLE_FAN);
11        glVertex2f(arrayX[j], arrayY[j]);
12        const int numSegments = 16;
13        for (int i = 0; i <= numSegments; i++)
14        {
15            float angle = i * 2.0f * 3.1415926f / numSegments;
16            float dx = radius * cosf(angle);
17            float dy = radius * sinf(angle);
18            glVertex2f(arrayX[j] + dx, arrayY[j] + dy);
19        }
20        glEnd();
21    }
22
23    glutSwapBuffers();
24 }

```

Tras el desarrollo del programa secuencial, fue necesario implementar paralelismo con el fin de mejorar el rendimiento y la eficiencia del procesamiento de los datos, ya que, al ser eventos computacionalmente demandantes, se buscó reducir la velocidad en que los datos se procesan. En concreto, se decidió realizar paralelismo de tareas. La forma en que se decidió aplicar la distribución de tareas fue en una parte del código. Se implementó paralelismo de tareas en la función de **update**, que como se mencionó previamente, es la función en donde se iteran todas las esferas, y se les actualizan sus posiciones en el eje horizontal y vertical, y se modifican sus velocidades y aceleraciones. Con esto logramos que se distribuyan las creaciones de las esferas y modificaciones de sus atributos, con el fin de acabar con estos procesos lo antes posible aprovechando todos los hilos que se poseen.

```

1 void update(int value)
2 {
3     double start_time = omp_get_wtime();
4     #pragma omp parallel for num_threads(thread_num)
5     for (int i = 0; i < alfaParticle.size(); i++)

```


Conclusiones

- Como era de esperarse, el tiempo correspondiente a los cálculos realizados en la función de update, los cuales se hacen cada instante, fue reducido una vez se implementó el paralelismo de tareas. Esto se debe a que se logró dividir las tareas a realizar en varios hilos con el fin de que cada uno desarrolle una sub-tarea y no llevar una cola secuencial de las mismas.
- La mejora de rendimiento del programa paralelo con respecto al secuencial no fue tan notoriamente positiva. Esto se debe a que los cálculos que realiza el screensaver desarrollado no son tan computacionalmente costosos, puesto que aparte de cálculo de senos y cosenos, se realizan las cuatro operaciones aritméticas básicas.

Recomendaciones

- Identificar las secciones de código donde sea útil y beneficioso utilizar paralelismo, utilizando las directivas y las cláusulas adecuadas para poder obtener el mayor rendimiento y aprovechar al máximo los recursos de la computadora.
- Evitar el uso de clases dentro de la estructura del programa, esto con el fin de evitar darle trabajo extra al programa y que pueda producir mejores resultados en cuanto a tiempo y performance se refiere.
- Cuando se necesite desarrollar un programa computacionalmente demandante, es recomendable implementar paralelismo en él con el fin de tener un procesamiento de tareas más rápido y más eficiente y así ofrecer la mejor experiencia de usuario posible.
- Si se desea desarrollar un programa que requiera el desarrollo gráficos en dos o tres dimensiones, se recomienda utilizar la librería de ya sea OpenGL o GLUT, el cual es una implementación de OpenGL. Esto ya que, en comparación de otras librerías, OpenGL/GLUT es una librería fácil de instalar, es compatible con numerosos sistemas operativos, está diseñado para aprovechar el hardware de un equipo de la mejor manera posible, es altamente flexible y configurable para satisfacer las necesidades y requerimientos del usuario y tiene su versión en una gran diversidad de lenguajes de programación.

Apéndice

Instalación de un compilador de C en Windows

A continuación, se proporcionan instrucciones paso a paso para instalar el compilador de C utilizando MSYS2 en un sistema operativo Windows:

1. Descargar e instalar MSYS2 desde el sitio web oficial (<https://www.msys2.org/>).

2. Abrir la consola de MSYS2. Para hacer esto, haz clic en el botón de inicio de Windows, escribir "MSYS2" y hacer click en "MSYS2 MSYS".
 3. Actualizar los paquetes existentes ejecutando el siguiente comando en la consola:
`pacman -Syu.`
 4. Instalar el compilador de C ejecutando el siguiente comando en la consola:
`pacman -S mingw-w64-x86_64-toolchain.`
 5. Verifica que el compilador de C esté instalado correctamente ejecutando el siguiente comando en la consola:
`gcc --version.`
Debería de verse la versión del compilador de C instalado en el sistema.
 6. Ya está listo para compilar y ejecutar programas en lenguaje C
- (Amin, 2023)

Instalación de OpenGL en Windows

A continuación, se proporcionan instrucciones paso a paso para instalar OpenGL en Windows para utilizarlo en programas en C y C++:

1. Descargar e instalar el kit de desarrollo de software de OpenGL. El SDK de OpenGL se puede descargar desde el sitio web oficial de OpenGL (<https://www.opengl.org/sdk/>).
2. Extraer los archivos del SDK en una carpeta en tu disco duro.
3. Agrega la ruta de la carpeta del SDK de OpenGL a la variable de entorno PATH.
4. Descarga e instala un compilador de C o C++
5. Agregar las librerías en las importaciones
6. Compilar y ejecutar el programa

Bibliografía

Powers, J. G., Klemp, J. B., Skamarock, W. C., Davis, C. A., Dudhia, J., Gill, D. O., ... Duda, M. G. (2017). The Weather Research and Forecasting Model: Overview, System Efforts, and Future Directions. *Bulletin of the American Meteorological Society*, 98(8), 1717–1737. doi:10.1175/bams-d-15-00308.1

Chapman, B., Jost, G., & van der Pas, R. (2008). *Using OpenMP: Portable Shared Memory Parallel Programming* (2nd ed.). The MIT Press

Markoff, J. (1990, June 29). Screensavers: A P.C. Fad That's Forever Changing. *The New York Times*.

Amin, M. F. B. (2023, 14 marzo). How to Install C and C++ Compilers on Windows. [freeCodeCamp.org](https://www.freecodecamp.org/news/how-to-install-c-and-cpp-compiler-on-windows/).
<https://www.freecodecamp.org/news/how-to-install-c-and-cpp-compiler-on-windows/>

Heller, M. (2022, 16 septiembre). What is CUDA? Parallel programming for GPUs. InfoWorld.

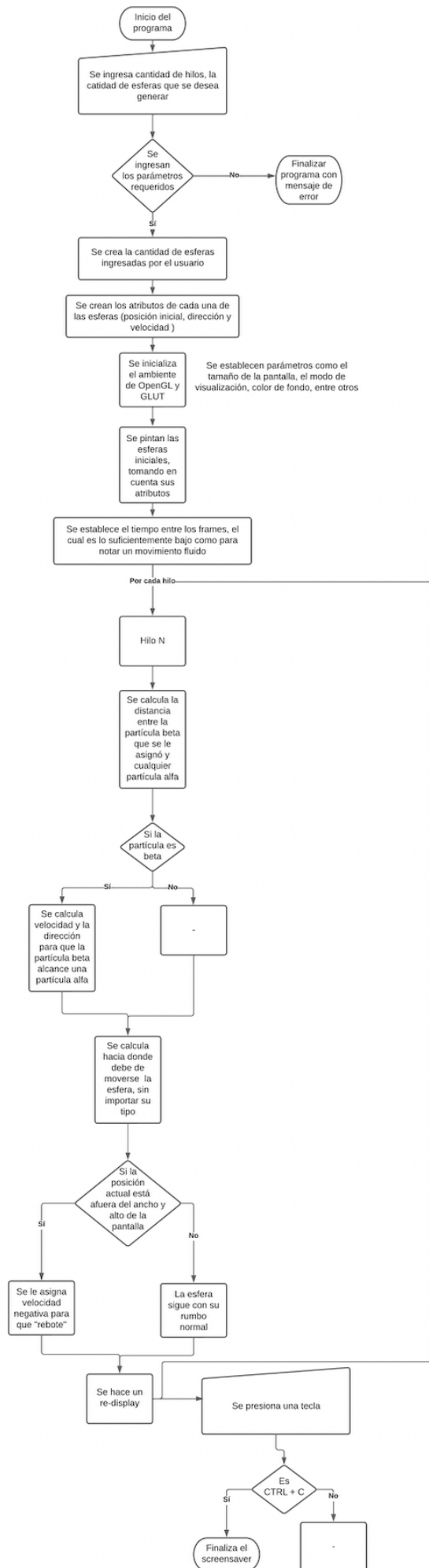
<https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html>

What is the Purpose of Screen Savers? (2023, 3 marzo). Easy Tech Junkie.

<https://www.easytechjunkie.com/what-is-the-purpose-of-screen-savers.htm>

Anexos

Diagrama de flujo



Instrucciones para correr los programas

Para el programa paralelo:

```
g ++ -o main2p main_parallel.cpp -lGL -lGLU -lglut -fopenmp
```

```
./main2p cantidadParticulasAlfa cantidadParticulasBeta cantidadHilo
```

Tipo de *cantidadParticulasAlfa*: entero

Tipo de *cantidadParticulasBeta*: entero

Tipo de *cantidadHilos*: entero

Para el programa secuencial:

```
g ++ -o main2p main_parallel.cpp -lGL -lGLU -lglut -fopenmp
```

```
./main2p cantidadParticulasAlfa cantidadParticulasBeta
```

Tipo de *cantidadParticulasAlfa*: entero

Tipo de *cantidadParticulasBeta*: entero

Catálogo de Funciones

Tabla #1: Catálogo de funciones

Nombre de la función	Entrada	Salida	Descripción
glutInit	<ul style="list-style-type: none">• Puntero de la variable sin modificar de argc• Puntero de la variable sin modificar de argv	void	Se utiliza para indicar la inicialización de la librería GLUT
glutInitDisplayMode	<ul style="list-style-type: none">• Una constante predefinida en la librería GLUT que representa un entero, el cual indica el modo de	void	Se utiliza para definir el modo de visualización de la ventana en la que se va a trabajar.

	visualización		
glutInitWindowSize	<ul style="list-style-type: none"> Ancho de la ventana a generar (int) Alto de la ventana a generar (int) 	void	Se utiliza para definir el ancho y el alto de la pantalla mediante dos enteros que representan valores en píxeles
glutCreateWindow	<ul style="list-style-type: none"> Nombre de la ventana, la cual es una cadena de texto 	Una ventana del tamaño indicado	Se utiliza para generar una ventana del tamaño indicado en glutInitWindowSize
glMatrixMode	<ul style="list-style-type: none"> GLenum mode: Especifica qué pila de matriz es el destino de las operaciones de matriz posteriores. 	void	especifica qué matriz es la matriz actual
glLoadIdentity	N.A.	void	Reemplaza la matriz actual con la matriz identidad
glOrtho	<ul style="list-style-type: none"> GLdouble left: posición del borde izquierdo de la ventana GLdouble right: posición del borde derecho de la ventana GLdouble bottom: posición del borde inferior de la ventana GLdouble top: posición del borde superior de la ventana GLdouble nearVal: 	void	Se utiliza para establecer una matriz de proyección ortográfica

	<p>Especifica las distancia a la profundidad más cercana</p> <ul style="list-style-type: none"> GLdouble farVal: Especifica las distancia a la profundidad más lejana 		
glClearColor	<ul style="list-style-type: none"> GLfloat red: valor del componente rojo GLfloat green: valor del componente verde GLfloat blue: valor del componente azul GLfloat alpha: valor de transparencia 	void	Se utiliza para especificar el color de fondo que se utilizará para borrar el búfer de color de la ventana de visualización.
glutDisplayFunc	<ul style="list-style-type: none"> func f : La nueva función de devolución de llamada de pantalla 	void	Establece la devolución de llamada de visualización para la ventana actual.
glutTimerFunc	<ul style="list-style-type: none"> int msec: número en milisegundos 	void	Registra un callback con un temporizador para que se active en un número específico de milisegundos
glutMainLoop	N.A.	void	Inicia el ciclo de procesamiento de GLUT
update	N.A.	Los cálculos de distancia entre una esfera y otra, la velocidad y la	Es la función en donde se hacen los cálculos necesarios para representar las

		aceleración de cada esfera, los cuales se almacenan en variables globales.	animaciones que se basan en la velocidad, aceleración y distancia de las esferas.
display	N.A.	Un frame correspondiente a una animación que se actualiza cada 16 milisegundos	Es la función en donde se toman los cálculos desarrollados en update para representar las distintas animaciones en pantalla.

Bitácora de Pruebas

Cálculo de speedup y efficiency

Tabla #2: Resultado de tiempos secuenciales, paralelos y speedup

SECUENCIAL (Tiempo en Segundos)	PARALELO (Tiempo en Segundos)	SPEEDUP
0.049	0.042	1.166666667
0.0457	0.0132	3.462121212
0.0474	0.0097	4.886597938
0.0448	0.0094	4.765957447
0.0472	0.0098	4.816326531
0.0464	0.0106	4.377358491
0.0488	0.0093	5.247311828
0.0482	0.0095	5.073684211
0.0486	0.0134	3.626865672
0.0482	0.0127	3.795275591

Speedup Promedio: 4.121816559

Efficiency: $Speedup/threads = 4.121816559/6 = 0.686969426$

El valor obtenido de speedup muestra una mejora de 4 veces en la ejecución paralela en comparación con la ejecución secuencial, esto muestra que al utilizar la librería OpenMP y sus directivas, se puede ver una mejora considerable tanto teóricamente como visualmente, al verse un incremento en los fotogramas por segundo en pantalla. Cabe mencionar que para el cálculo de speedup del programa paralelo se utilizó el promedio de los tiempos de ejecución secuencial con el mismo valor de N. Estos fueron medidos en las tablas anteriores.

En cuanto al valor obtenido al calcular la eficiencia indica que el programa está aprovechando aproximadamente el 68.7% de los recursos disponibles al ejecutarse en paralelo con un 6 threads, por lo tanto la capacidad del programa para escalar de manera efectiva en diferentes sistemas paralelos se considera buena.

Capturas de los tiempos obtenidos en la ejecución del programa

Captura de tiempo paralelo

```
Elapsed time: 0.0102 seconds
Elapsed time: 0.0153 seconds
Elapsed time: 0.0105 seconds
Elapsed time: 0.0157 seconds
Elapsed time: 0.0143 seconds
Elapsed time: 0.0173 seconds
Elapsed time: 0.0171 seconds
FPS: 61.75
Elapsed time: 0.0098 seconds
Elapsed time: 0.0117 seconds
Elapsed time: 0.0100 seconds
Elapsed time: 0.0149 seconds
Elapsed time: 0.0167 seconds
Elapsed time: 0.0106 seconds
Elapsed time: 0.0100 seconds
Elapsed time: 0.0100 seconds
Elapsed time: 0.0111 seconds
```

Captura de tiempo secuencial

```
Elapsed time: 0.0469 seconds
Elapsed time: 0.0501 seconds
Elapsed time: 0.0472 seconds
Elapsed time: 0.0488 seconds
Elapsed time: 0.0458 seconds
Elapsed time: 0.0467 seconds
FPS: 20.59
Elapsed time: 0.0458 seconds
Elapsed time: 0.0471 seconds
Elapsed time: 0.0461 seconds
Elapsed time: 0.0473 seconds
Elapsed time: 0.0471 seconds
Elapsed time: 0.0472 seconds
Elapsed time: 0.0477 seconds
Elapsed time: 0.0478 seconds
Elapsed time: 0.0488 seconds
Elapsed time: 0.0460 seconds
```

Enlace al repositorio

<https://github.com/oscarparedez/proyecto1-paralela>